MASTER EN SCIENCES INFORMATIQUES

Elaboration d'une plateforme orientée objet dédiée à la coregistration d'images médicales

Vaesen, Olivier

Award date:
2003

Facultés Universitaires Notre-Dame de la Paix
Namur
Institut d'Informatique

Elaboration d'une plateforme orientée objet dédiée à la coregistration d'images médicales

Olivier Vaesen

Mémoire présenté en vue
de l'obtention du grade
de Maître en Informatique

Promoteur
Professeur J-P Leclercq
Co-promoteur
H. Meurisse

Année académique 2002-2003

Résumé Etablir un diagnostique sur base de clichés issus de modalités différentes n'est pas aisé. En effet, ces images doivent souvent être recalées afin de faire correspondre les organes représentés sur chacune d'entre elles. Ce recalage fait appel à un ensemble de transformations géométriques, complexes pour l'homme. Des algorithmes ont été créés pour résoudre ce type de problèmes. Ce mémoire ne s'attache pas à définir une nouvelle méthode de recalage, mais décrit l'architecture d'une plateforme capable d'accueillir en son sein tout algorithme de coregistration, via une modélisation générique de quatre composantes dont tout algorithme de recalage est constitué : l'espace des caractéristiques, la transformation, la fonctionnelle d'appariement et la méthode d'optimisation. Nous commençons par décrire les notions de base pour comprendre au mieux le problème de la coregistration. Ensuite, nous abordons chacune de ces quatre composantes. Pour illustrer nos propos, une étude de cas portant sur un programme issu d'*AIR*, un package d'outils dédié à la coregistration d'images médicales, est réalisée. Elle consiste en l'identification de ces quatre composantes, en vue de l'insertion au sein de la plateforme. Par après, nous décrivons l'ensemble de l'architecture du programme. Finalement, nous critiquons l'état de développement et proposons des perspectives quant à l'évolution de la plateforme.

Abstract To make out a diagnosis on the basis of negatives from different modalities is not easy. Indeed, these images must often be realigned, so that the organs represented on each of them correspond to each other. This realignment calls for geometrical transformations, complex for humans. Algorithms have been developed in order to resolve that kind of problems. This thesis doesn't attempt to define a new realignment method, but describes the architecture of a plateform wich supports any coregistration algorithm, with the help of a generic modelisation of four components which any realignment algorithm is made of : the feature's space, the transformation, the pairing function and the optimisation method. We start by describing the basic notions in order to clearly understand the coregistration problem. Then we propose an overview of each of these four components. To illustrate our words, we realize a study case on a program featured in AIR, a librairy wich is dedicated to the coregistration of medical images. It consists in identifying these four components, in order to integrate them in the platform. Afterwards, we describe the overall program architecture. Finally, we express some critics about the development state of the solution and propose evolution perspectives for it.

Je tiens à remercier mon promoteur, le professeur Jean-Paul Leclerq, pour ses conseils et le suivi qu'il m'accordé tout au long de la réalisation de ce mémoire.

Je tiens également à exprimer ma reconnaissance à mon co-promoteur et maître de stage, Hubert Meurisse, qui m'a guidé tout au long de l'implémentation de ce travail. Je le remercie également pour les conseils qu'il m'a donnés lors de la rédaction de ce mémoire.

Je tiens également à remercier le personnel du département de médecine nucléaire des Cliniques Universitaires de Mont-Godinne. Je me souviendrais de quelques bons moments passés en leur compagnie.

Il m'est impossible de ne pas citer Bruno, aussi précieux pour ses conseils que pour la bonne humeur qu'il sait insuffler.

Merci aussi à Isabelle et Pierre pour leur aide tant pour la mise en page de ce mémoire que pour des questions de fond. Je tiens également à les remercier pour ces nombreuses "soirées détentes" passées en leur compagnie.

Je tiens encore à remercier tous les gens qui m'ont permis de passer les cinq années les plus formidables de ma vie. Je pense aux Dimitri, aux Len, à Mim, Sabrina, Rico, Butch, Yann, Tom, Laurent, Beben , Francesco, "la bande du rézo"... Ils sont trop nombreux à citer mais je pense à vous quand j'écris ces lignes.

Comment ne pas remercier Papa et Maman pour leur soutien tout au long de ces cinq années, vous m'avez aidé dans les moments difficiles, vous m'avez supporté aussi. Un petit clin d'oeil aussi à mon petit frère. C'est à ton tour maintenant !

A tous, merci.

Table des matières

Introduction

Lorsqu'il est difficile pour un docteur d'établir un diagnostique de prime abord, le patient est souvent convié à passer des examens radiologiques, tels que des tomographies. Pour ce faire, différentes approches (ou modalités) peuvent être envisagées, chacune permettant de saisir un aspect de l'organisme. Selon les cas, une ou plusieurs de ces approches doivent être considérées pour que les médecins puissent émettre un diagnostique.

Généralement, chaque examen rend un ensemble de clichés où chacun d'entre-eux représente une tranche du volume relatif à la zone concernée par l'examen. La technique "classique" utilisée pour établir un diagnostique consiste à consulter, un par un, chacun de ces clichés. Dans l'hypothèse où le médecin ne doit se baser que sur un seul examen, sa tâche se résume alors à analyser les clichés proprement dits. Par contre, lorsqu'il doit se baser sur plusieurs séries de clichés issues de diverses modalités, son travail se complique. En effet, il faut savoir qu'en fonction de l'approche choisie, les images sont très différentes, tant sur le plan du contenu que sur le plan du format. Deux images pourront par exemple avoir des rendus visuels complètement différents, varier au niveau du nombre de tranches formant le volume ou encore rendre compte d'organes identiques mais orientés totalement différemment. Le médecin devra donc fournir des efforts cognitifs importants pour faire correspondre les organes considérés et parvenir ainsi à établir un diagnostique de la situation. Ce problème de réalignement d'images est connu sous le nom de coregistration.

Des outils informatiques ont été mis au point pour aider le médecin dans cette tâche. En effet, il existe aujourd'hui différents programmes et packages d'outils dédiés à ce type de problèmes. Chacun d'entre eux possède ses propres particularités, ses avantages et inconvénients. En sachant cela, il serait pertinent de pouvoir proposer au médecin une plateforme capable d'intégrer chacun de ces outils. Il est même possible d'aller plus loin. Stéphane Seynave [Sey02] a montré dans son mémoire que tout algorithme de coregistration, quelqu'il soit, peut être décomposé en quatre composantes. Si l'on parvient à modéliser ces composantes de manière suffisamment générique, alors il est possible non seulement d'intégrer ces algorithmes au sein d'une plateforme mais également de remplacer certaines composantes, au sein d'un algorithme, par d'autres. Cette optique permettrait de fournir un éventail très large de méthodes, de manière à adapter au mieux

l'algorithme en fonction des images à coregistrer.

Ce mémoire est consacré à la description et à l'implémentation d'une plateforme orientée objet dédiée à la coregistration d'images médicales.

Pour comprendre la modélisation effectuée, il convient tout d'abord de poser ou rappeler quelques notions de base. Cela fera l'objet du chapitre premier. Après un bref historique, nous reviendrons sur le concept d'image numérique que nous étendrons à la notion d'image médicale. Nous aborderons ensuite les différents types d'images médicales. Cette section devrait permettre au lecteur de sentir plus précisément les problèmes liés à la coregistration. Nous parlerons ensuite de la présentation des résultats de la coregistration, très importante car elle doit permettre l'élaboration d'un diagnostique. Nous achèverons ce chapitre par une approche intuitive des quatre composantes communes à tous les algorithmes.

Le deuxième chapitre sera consacré à la présentation de chacune de ces composantes. Chacune d'entre elles sera abordée suffisamment pour comprendre la modélisation qui en a été faite au sein de la plateforme. Cette partie constitue en effet une des clés de ce programme.

Le troisième chapitre annoncera les technologies utilisées dans la réalisation de la plateforme. On y abordera les différents langages et techniques de programmation utilisés, ainsi que les divers formats et outils concernés.

Le quatrième chapitre sera l'objet d'une étude de cas mettant en application les concepts explicités au deuxième chapitre. Le package *AIR*, librairie open source (dont les sources sont publiques) d'outils dédiés à la coregistration d'images médicales, y sera analysé.

Le cinquième chapitre abordera quant à lui la modélisation et l'implémentation de la plateforme. Ses divers constituants y seront explicités. Nous parlerons également quelque peu des outils auxiliaires développés pour le programme.

Enfin, le sixième chapitre critiquera l'état actuel de développement de la plateforme. Il mettra en évidence ses faiblesses et défauts, et proposera quelques modifications pour en améliorer le fonctionnement.

Chapitre 1

Coregistration d'images médicales : contexte et notions fondamentales

1.1 Historique et intérêt

"Voir à l'intérieur du corps humain sans nuire"
Hippocrate (Cos, 460 - Thessalie, 377 av. J.-C.)

Il y a quelque deux mille cinq cents ans, Hippocrate formulait ce rêve. Pouvoir visualiser l'intérieur d'un corps sans devoir y toucher. Il aura fallu attendre plus de deux mille ans pour voir son vieux rêve enfin se réaliser.

On considère que le père de la radiologie est le physicien allemand Whilem Conrad Röntgen. Un après-midi de novembre 1895, alors qu'il tentait de reproduire une expérience pour étudier les tubes cathodiques et le phénomène de luminescence, il observa un rayonnement inconnu. Il prit tout d'abord soin d'envelopper un tube cathodique dans du carton opaque pour qu'il ne soit pas gêné par la lumière engendrée à l'intérieur du tube. Il le met alors sous tension et, finalement, plonge la pièce dans l'obscurité. A sa grande satisfaction, aucune lumière ne passe au travers de l'enveloppe opaque. Alors qu'il allait débrancher le tube, il remarque qu'au fond du laboratoire, l'écran qui allait lui servir pour l'expérience scintille légèrement. Il allume et éteint le tube successivement. Chaque fois que le tube est sous tension, l'écran s'illumine. C'est alors que Röntgen a le trait de génie de penser que la fluorescence n'est pas provoquée par les rayonnements cathodiques mais par des rayonnements inconnus, générés secondairement par ces rayonnements cathodiques. Si ces rayonnements sont capables de passer au travers du carton opaque,

peut-être sont-ils capables de traverser d'autres substances. Röntgen met sa main entre le tube et l'écran. Ce sont ses os qui apparaissent. Les rayons X étaient découverts.

Le vingtième siècle a vu l'émergence de nouvelles techniques d'imagerie que l'on peut sans tort qualifier de révolutionnaires. Elle ont non seulement permis d'appréhender le corps dans sa géométrie naturelle, c'est à dire en trois dimensions, mais sont parvenues à le faire de manière quasiment non-invasive.

Enormément d'images médicales sont quotidiennement acquises dans le monde. Une fois acquises, ces images doivent encore être examinées et interprétées par le médecin. Bien souvent, celui-ci réalise l'examen à partir d'une succession de coupes en deux dimensions. On perçoit bien évidemment la difficulté de percevoir les objets dans leur globalité, d'autant plus lorsque des images doivent être étudiées simultanément.

Une des tâches courantes du praticien est la comparaison d'images d'un même patient, acquises à des moments différents ou bien issues d'examens différents. Ces tâches relèvent toutes de la même problématique : le recalage. Celui-ci consiste à déterminer une relation géométrique entre deux objets représentés par des images. Ces relations sont peu aisées à déterminer. Les recherches se sont donc évidemment portées vers la conception de méthodes automatiques ou semi-automatiques de recalage. Des packages plus ou moins efficaces ont vu le jour. On citera par exemple *AIR* (Automated Image Registration), *SPM* (Statistical Parametric Mapping) ou encore *mmvreg* (Multi Modality Volume Registration). Ces méthodes affichent plus ou moins de performances (et de résulats) selon les situations dans lesquelles elles se trouvent. Par situations, on peut penser aux types d'images médicales que ces méthodes doivent traiter, aux caractéristiques de l'image. Pourquoi ne combinerions-nous pas les différents avantages des méthodes existantes, en choisissant les méthodes fournissant les meilleurs résultats selon la situation rencontrée ?

Avant d'entrer directement au coeur des processus de coregistration, il est important de mettre en place les notions sur lesquelles nous reviendrons régulièrement. Dans un contexte de coregistration d'images médicales, il est évident qu'il faille revenir sur la nature même d'image médicale. Nous reviendrons aussi sur la notion de modalité, qui est la source du problème qui nous concernera tout au long de ce travail.

1.2 Images médicales

Cette section est destinée à mettre en place le concept d'image médicale, bien plus riche qu'une "simple image". Pour ce faire, nous rappelerons dans un premier temps le concept d'image numérique ainsi que ses divers attributs. Nous nous attacherons ensuite à enrichir l'image de concepts tels qu'elle puisse trouver son sens dans le domaine médical.

1.2.1 L'image numérique

Lorsque l'on parle du concept d'image médicale, la première "image" qui nous vient à l'esprit est celle d'un cliché représentant une certaine portion du corps. Bien que, comme nous le verrons par la suite, une image médicale ne peut se résumer à cette dimension, ces clichés en font indéniablement partie. Nous allons décrire ce qu'est une image numérique, afin de comprendre au mieux la modélisation qui en sera faite lors de la description de l'architecture de la plateforme.

Traditionnellement, on distingue deux types d'images numériques : les images *vectorielles* et les images *matricielles.*

Dans une image vectorielle, les données sont représentées par des formes géométriques simples qui sont décrites d'un point de vue mathématique. Le dessin est alors mémorisé comme "tracer une droite entre (x_1, y_1) et (x_2, y_2), puis tracer un cercle de centre (x, y) et de rayon z". Nous n'entrerons pas plus dans les détails dans la description de ce type d'image car celui-ci sort du cadre d'utilisation qui nous intéresse.

Une image est dite matricielle si elle est formée d'un tableau de points (appelés **pixels**). En fonction de la valeur scalaire attribuée à chacun des points du tableau, l'image prendra forme. Nous partirons du concept classique d'image en deux dimensions pour introduire les concepts de base. Cette image bi-dimensionnelle peut être considérée comme une fonction à deux arguments qui, pour tout couple (x, y) représentant des coordonnées spatiales, renvoit une valeur scalaire du tableau.

Dans le domaine d'application qui nous intéresse, les images ne sont pas seulement bi-dimensionnelles. Elles peuvent être également tri-dimensionnelles[1], quadri-dimensionnelles[2] et même plus ! C'est la raison pour laquelle nous devons étendre le concept de pixel aux images en trois dimensions. Ces images 3D sont en fait constituées d'une série de tranches d'images 2D, d'épaisseur bien définie. On ne peut plus a priori parler de pixel dans ce cas, mais on nommera alors **voxel** le volume élémentaire correspondant dans la tranche au pixel de l'image 2D. Nous avons jusqu'à présent omis la notion de dimension et de résolution de l'image. Nous l'abordons dans le paragraphe suivant.

[1] les trois dimensions spatiales

[2] les trois dimensions spatiales et une dimension temporelle

1.2.2 Résolution

Si un pixel[3] représente "un point" de l'image, il n'en demeure pas moins qu'on puisse lui affecter certaines dimensions. Il est important de distinguer deux notions : la *résolution* d'une image et la *taille* d'une image.

La résolution d'une image est le nombre de pixels par unité de longueur utilisés pour définir cette image. Plus la taille des pixels est petite, plus la résolution de l'image est meilleure et plus l'information sur l'objet observé est fidèle.

La taille d'une image, quant à elle, indique simplement le nombre de pixels dont elle est constituée. Une image IRM aura par exemple une taille de 192 x 256 x 20, ce qui signifie que l'image 3D sera composée de 20 tranches, et que ces dernières seront elles-mêmes composées de 192 colonnes et 256 lignes.

1.2.3 Codage

Pour qu'une image puisse rendre une information, il faut qu'elle puisse présenter un contraste de couleurs suffisant (on parle aussi de palette de couleurs ou encore d'échelle de couleurs). Ce contraste de couleurs s'exprime au niveau du nombre de bits nécessaires à l'encodage de chaque pixel. Généralement, on reconnaît que l'oeil humain est capable de discerner 300 000 couleurs. Une profondeur d'encodage de 24 bits par pixel (ou quelque 16 millions de couleurs) semble donc tout à fait adéquate pour rendre n'importe quelle image.

Cependant, en imagerie médicale, il est courant de voir des pixels encodés sur 32 bits. Cela est dû à la nature même des images. En effet, chaque image est le fruit d'une acquisition par du matériel médical perfectionné. Ces matériels relèvent des grandeurs physiques avec une extrême précision. Ces grandeurs peuvent aussi bien être des nombres entiers que des nombres réels. L'échelle des valeurs que les machines recueillent peut être telle qu'elle doive être codée sur 32 bits par pixel. De par cette "nature physique" des images médicales, il est un fait que les valeurs ne correspondent pas vraiment à des valeurs de rendu classique, contrairement à une photographie . Pour rendre de telles valeurs physiques, il faut utiliser un procédé de projection des valeurs dans un espace de couleurs. Ainsi, si l'on désire rendre une image codée sur 12 bits sur échelle de 256 niveaux de gris, chaque couleur rendue ne correspondra pas à une valeur unique, mais bien à une plage de valeur. Dans l'exemple, chaque pixel représentera une tranche de 16

[3] ce que nous dirons à présent pour les pixels sera également valable pour les voxels

valeurs[4].

1.2.4 Le contexte médical

Les deux sections précédentes ont discuté de notions techniques. Ce paragraphe va s'intéresser aux attributs inhérents aux images du monde médical, de telle sorte que celles-ci puissent réellement avoir un sens.

Ce sens s'exprime au travers d'informations qui sont contenues dans ce que l'on pourrait nommer le contexte médical de l'image ou encore son environnement. Il s'agit de données administratives, nécessaires à la bonne interprétation de l'image (ce qui revêt, bien entendu, une importance capitale). On peut donc voir l'image médicale comme étant composée de deux grandes parties, la première étant les données brutes (le tableau de pixels), la seconde, son contexte.

Nous allons à présent décrire brièvement les classes d'informations que l'on retrouve dans la partie contextuelle de l'image. Nous distinguerons quatre grandes classes : les informations générales, communes à toutes les images médicales, les informations concernant la troisième dimension spatiale, les informations relatives à la dimension temporelle et enfin les informations concernant la modalité utilisée. Nous ne détaillerons pas l'ensemble de toutes les données présentes, nous essaierons plutôt de faire sentir au lecteur les concepts sous-jacents à chaque classe d'informations.

1.2.4.1 Les informations générales

Nous regroupons dans cette classe les informations que toute image médicale possède, quelque soit sa nature. On y distingue les **informations techniques de base**, telles que la taille des pixels, le nombre de colonnes, de lignes, le nombre de bits sur lequel chaque pixel est encodé, des valeurs remarquables, ... On peut également y trouver les **données relatives au patient**, telles que son nom, son âge ou un ID. On y trouvera également des informations concernant l'**examen** (date, numéro d'examen, ...) et les **séries** d'images (numéro identifiant, le physicien qui a mené l'examen, ...).

[4]Nous avons considéré ici une projection homogène. Rien ne nous empêche d'envisager des projections qui ne le seraient pas.

1.2.4.2 La troisième dimension

Dans le cas d'images médicales tri-dimensionnelles ou de dimension supérieure, des informations supplémentaires sont nécessaires pour les caractériser. On pensera à l'épaisseur des tranches, à l'écart entre les tranches (il arrive souvent qu'il existe "un trou" entre la fin d'une tranche et le début de la suivante) ou encore à l'ordre des tranches.

1.2.4.3 La dimension temporelle

La dimension temporelle regroupe les informations relatives à la dynamique de l'étude. On y retrouve principalement le nombre d'images en trois dimensions acquises lors de l'examen, le temps au début de l'examen et à la fin.

1.2.4.4 La modalité

Le concept de modalité faisant l'objet de la section suivante, nous n'en dirons pas plus dans ce paragraphe. On trouvera simplement dans cette classe des informations relatives à la modalité employée.

1.3 Modalités

Nous allons dans un premier temps définir ce qu'est une modalité. Ensuite, nous discuterons des différents types de modalités, en les classant en deux familles, selon le type d'images qu'elles permettent de rendre. J'illustrerai chacune de ces deux familles par une modalité particulière. Enfin, nous exprimerons quelques difficultés quant à une éventuelle mise en correspondance des clichés provenant de modalités différentes.

1.3.1 Définition

Depuis la découverte des rayons X en 1895 par le physicien allemand Whilheim Conrad Röntgen, les techniques en matière d'imagerie médicale ont évidemment énormément évolué. Ces méthodes d'acquisition portent le nom de *modalités*. Elles désignent à la fois le protocole clinique et la technique utilisée pour produire une image d'une propriété tissulaire particulière. La scintigraphie, l'échographie et la résonance magnétique en sont des exemples. Etablir et décrire de manière exhaustive l'ensemble des modalités existantes

ne présente que peu d'intérêt dans le cadre de ce travail. Tout au plus, nous essaierons de classifier les modalités selon le type d'images qu'elles permettent d'appréhender.

On distinguera ainsi deux grandes familles d'images : les images anatomiques et les images fonctionnelles. Elles seront explicitées dans les paragraphes suivants et accompagnées d'une modalité afin de les illustrer[5].

1.3.2 Images anatomiques

Les images anatomiques (ou morphologiques) rendent compte de la structure interne de l'organisme. Elles permettent de visualiser le volume, la taille et la structure des tissus. L'IRM est un exemple de modalité rendant des images de ce type.

1.3.2.1 IRM - Imagerie par résonance magnétique

L'imagerie par résonance magnétique est basée sur le principe de la résonance des atomes de certaines molécules sous l'action d'ondes de radio-fréquences. L'appareil est constitué d'une sorte de tunnel formé d'un aimant très puissant, entourant le lit du patient (figure 1.1).

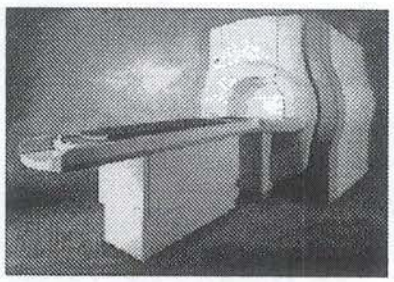


FIG. 1.1 – IRM ([Webd])

Des antennes émettent une onde qui excite les noyaux d'hydrogène contenus dans l'eau composant nos cellules[6]. Après cette brève stimulation (elle ne dure que quelques millisecondes), les atomes d'hydrogène restituent cette énergie dans différents plans de l'espace, sous l'influence du champ magnétique de l'aimant. Des antennes réceptrices

[5]Notons que certaines modalités permettent de rendre à la fois des images anatomiques et fonctionnelles.

[6]Pour rappel, près de 80% de notre corps est constitué d'eau.

captent alors cette énergie. Un ordinateur s'occupe finalement de reconstruire une véritable carte énergétique de la partie du corps concernée.

En fonction de la teneur en eau des différents tissus analysés, les images obtenues seront très différentes. Cette technique permet d'obtenir des images d'une très grande précision et des coupes dans tous les plans de l'espace. La figure 1.2 suivante illustre cette modalité.

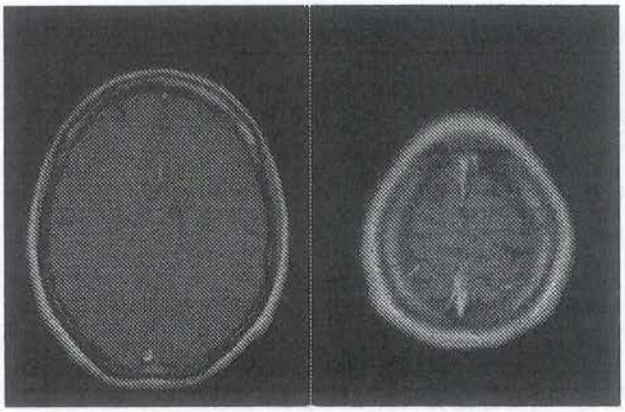


FIG. 1.2 – clichés issus d'une image IRM de la tête, constituée de 256 lignes sur 192 colonnes, en 20 tranches.

1.3.3 Imagerie fonctionnelle

Les images fonctionnelles permettent de mettre en évidence les activités métaboliques de notre corps, comme la perfusion d'un muscle. L'accent est donc mis sur les aspects dynamiques de l'organisme. La TEP rend de telles images.

1.3.3.1 TEP - Tomographie à émission de positons

La tomographie à émission de positons utilise une technique d'acquisition d'images médicales radicalement différente de celle utilisée par l'IRM. Le principe consiste à injecter dans le corps du patient des molécules biologiques marquées par des isotopes radioactifs[7]

[7]Pour un même élément chimique, il peut arriver que certains noyaux existent, différant uniquement par le nombre de neutrons. Ces noyaux sont dit isotopes. Certains noyaux radioactifs ont la particularité de se désintégrer en émettant des particules ou du rayonnement.

à demi-vie[8] très brève. Ces molécules sont appelées traceurs. Selon le traceur utilisé, une fonction de l'organisme va être explorée. La figure 1.3 montre un scanner TEP

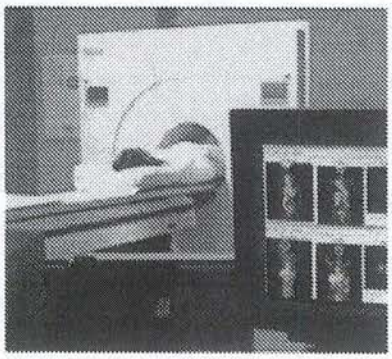


FIG. 1.3 – TEP ([Webd])

Le principe d'acquisition est lié à la désintégration de ces isotopes émetteurs de positons. Ils libèrent alors une quantité de rayonnement gamma. Cette quantité est proportionnelle à la concentration locale de l'isotope radioactif. Une caméra TEP permet de récupérer les photons gamma émis. Via un système photo-électrique, la reconstruction d'une image en trois dimensions est possible, via des algorithmes de reconstruction. La figure 1.4 montre une image TEP. On peut constater la différence de rendu par rapport à une image IRM.

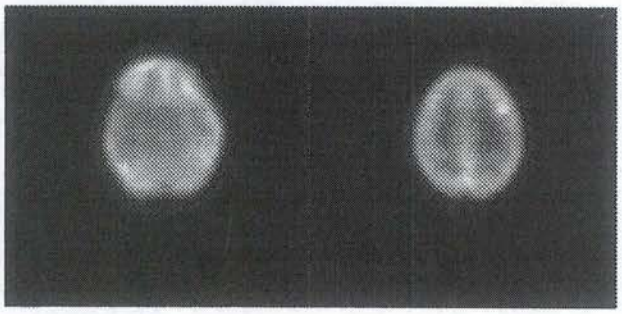


FIG. 1.4 – clichés issus d'une image TEP de la tête, en 128 lignes sur 128 colonnes, sur 63 tranches.

[8]La demi-vie est le temps au terme duquel une grandeur (physique, biologique) atteint la moitié de sa valeur initiale.

1.3.4 Correspondance des images de modalités différentes

D'un point de vue visuel, on peut observer une très grande diversité dans le rendu des clichés obtenus. Selon la nature des éléments observés, certains organes n'apparaîtront pas sur une modalité mais bien sur une autre. Pour reprendre les deux modalités citées plus haut, une IRM de la tête rendra compte de la partie osseuse de celle-ci alors qu'une TEP ne la montrera pas.

Des obstacles surviennent également lorsque l'on considère les attributs des images issues de modalités différentes. En effet, une image IRM ne proposera pas des dimensions équivalentes à une image TEP, telle que le montre les figures. En ajoutant à cela le fait que la taille des voxels varie selon la modalité, la mise en correspondance des clichés n'est que peu aisée.

Enfin, des obstacles supplémentaires apparaissent lorsque le patient passe d'un appareil à l'autre. En effet, la partie du corps soumise à l'examen se verra orientée d'une façon sur la première modalité et, peut-être, d'une autre sur la seconde. Tout dépend de la position du patient sur chacune d'elles.

Pour esquisser un diagnostique, le médecin devra fournir d'importants efforts mentaux pour essayer de faire correspondre différents clichés. Cette mise en correspondance, ou **coregistration**, sera abordée à la section suivante.

1.4 Coregistration

Maintenant que les notions de modalité et d'image médicale ont été posées, il convient de s'intéresser dorénavant au problème central de ce mémoire, à savoir la coregistration de deux images. Cette section abordera dans un premier temps la notion de coregistration et son intérêt. Ensuite, nous passerons volontairement la question du fonctionnement du processus de coregistration pour nous pencher sur la présentation d'images coregistrées. Enfin, nous tenterons de mettre en évidence certains aspects du processus de coregistration et ce, de manière intuitive.

1.4.1 Définition et intérêt

La coregistration (ou recalage) est un processus de réalignement de deux images, l'une étant appelée image-maître (ou image modèle) et l'autre image-esclave (ou image à recaler), chacune couvrant la même région du corps (dans le cas médical). L'image-esclave

va être modifiée par des translations, rotations et autres transformations, de telle sorte qu'elle corresponde au mieux à la première[9].

Bien sûr, on est en droit de se poser l'intérêt de cette pratique. Une esquisse de réponse a déjà été formulée dans la section précédente. Il est vrai que, dans certains examens cliniques, une seule modalité permet parfois de satisfaire à l'obtention de toutes les informations nécessaires au diagnostique. Cependant, il subsiste bien d'autres situations pour lesquelles il faut recourir à l'emploi de plusieurs modalités. Une fois les images en sa possession, le praticien devra les apparier, afin qu'il puisse les étudier.

Jusqu'à présent, nous avons abordé le problème de la coregistration sous un angle multimodal, c'est-à-dire en considérant l'appariement d'images provenant de modalités différentes. Pratiquement, ce n'est pas la seule situation pour laquelle la coregistration est interéssante. Ainsi, on pourra distinguer trois grandes classes de problèmes de coregistration en fonction du type d'images impliquées et de la comparaison entre individus différents ou non.

1.4.1.1 La coregistration intrasujet monomodale

Typiquement, ces problèmes concernent des pré-traitements relatifs à l'étude de séries temporelles d'images. On fait intervenir des données issues d'une même personne, prises dans des intervalles de temps importants, ou dans des intervalles de temps beaucoup plus courts, pour appréhender certains aspects dynamiques de l'organisme. On peut citer l'exemple d'IRM de patients atteints de sclérose en plaques, dont il est possible de suivre l'évolution par l'acquisition d'images échelonnées sur plusieurs mois. Un autre exemple est celui d'images fonctionnelles prises à quelques secondes d'intervalle, pour éventuellement détecter certaines activations neuronales.

1.4.1.2 La coregistration intrasujet multimodale

Il s'agit de la classe de problèmes que nous avons surtout évoqués jusque maintenant. Elle consiste à tirer profit d'informations complémentaires issues de modalités différentes. Trois scénarios peuvent se présenter, qu'il s'agisse de recaler

- deux images anatomiques
- une image fonctionnelle et une image anatomique
- deux images fonctionnelles

[9] Cette notion de correspondance sera analysée plus en profondeur dans le chapitre suivant

1.4.1.3 La coregistration intersujet

Il s'agit de comparer les clichés d'un individu avec ceux d'autres individus, afin de détecter d'éventuelles pathologies ou différences entre individus, par comparaison avec des individus sains ou avec des "atlas", qui sont en quelque sorte des modèles de référence.

Nous essaierons dans une section ultérieure de faire sentir intuitivement les divers mécanismes mis en oeuvre lors d'un recalage. Ceux-ci serviront de points de repère lorsque les caractéristiques des algorithmes de coregistration seront abordés, ce qui sera le cas au chapitre suivant. Avant cela, nous allons aborder un aspect très important lié au processus de coregistration : la présentation.

1.4.2 La présentation des résultats

Il faut tout d'abord se rappeler de l'intérêt premier de la coregistration : aider le médecin à analyser et diagnostiquer le patient. Pour ce faire, l'image-maître et l'image recalée doivent être présentées de telle manière que les informations cruciales soient mises en exergue. Il existe plusieurs façons de présenter ces résultats. On distingue deux grandes familles : l' affichage en parallèle (plus communément appellé *parallel display*) et l'affichage par superposition.

1.4.2.1 Parallel display

L'affichage en parallèle est une technique de visualisation dont le principe est assez simple, puisqu'il s'agit de présenter les images dans deux fenêtres adjacentes. Les deux images présentées correspondent à la même tranche. Le principal avantage de ce type de technique est qu'il ne modifie pas le contenu visuel des deux images. Néanmoins, il n'est pas aisé de faire coïncider mentalement les deux images.

On distingue plusieurs techniques d'affichage en parallèle. La plus basique est l'affichage en parallèle simple où les images sont simplement présentées de manière conjointe. La figure 1.5 illustre cette technique.

Cette technique de visualisation est relativement pauvre car elle demande un effort de le part de l'utilisateur pour faire coïncider les points des deux images. Des techniques de visualisation plus évoluées existent, telles que le *parallel display with link cursor.*

Cette méthode est similaire à la première, si ce n'est qu'il est possible de relier les

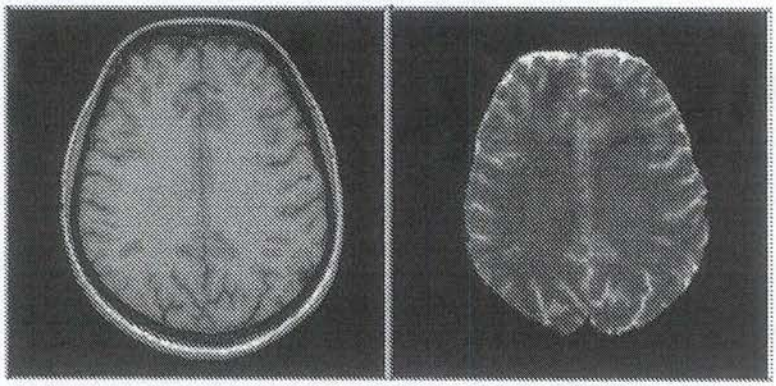


FIG. 1.5 – illustration de la technique de *parallel display simple* ([Webc]).

points d'une image aux points correspondants sur l'autre image. L'utilisateur peut ainsi mettre en évidence les points significatifs sur une image et retrouver leur correspondant sur l'autre. L'exemple de la figure 1.6 montre cette technique.

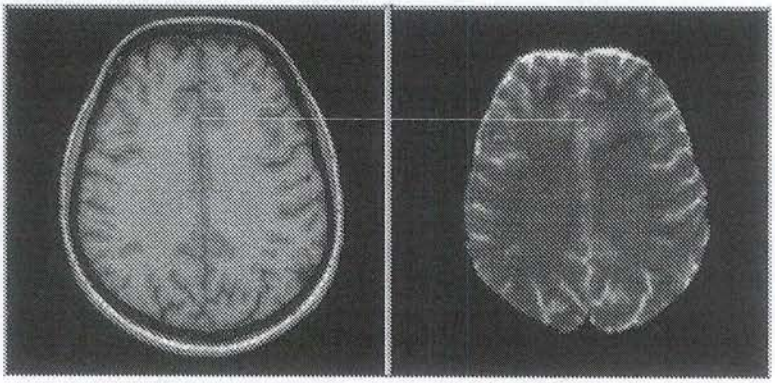


FIG. 1.6 – illustration de la technique de *parallel display with link cursor* ([Webc]).

On peut encore améliorer la technique de base. *Le parallel display avec mélange de parties d'images* permet à l'utilisateur de fondre une partie de l'image dans l'autre. Il peut ainsi sélectionner une zone d'une image et verra cette zone se dessiner sur l'autre image. La figure 1.7 illustre ce principe.

Bien que l'affichage en parallèle possède l'avantage de ne pas altérer les images, et donc leur contenu informationnel, ils présentent le désavantage d'exiger de la part de l'utilisateur un effort cognitif important étant donné qu'il doit transposer mentalement les points d'une image sur l'autre. Il doit en quelque sorte effectuer à nouveau un processus

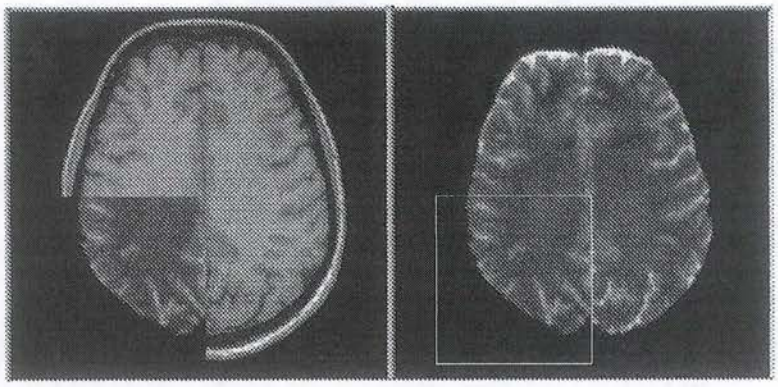


FIG. 1.7 – illustration de la technique de *parallel display avec mélange de parties d'images*([Webc]).

de coregistration ! Dans le cas où les techniques de visualisation modifient le contenu visuel des images (comme les techniques que nous verrons ci-dessous), le parallel display peut s'avérer une solution tout à fait honorable pour retrouver ces informations originelles.

1.4.2.2 La présentation par superposition

Les techniques de présentation par superposition diffèrent essentiellement par le fait qu'elles ne demandent pas d'efforts cognitifs pour évaluer la correspondance des deux images. Les techniques sont nombreuses ; nous en aborderons brièvement deux, à savoir *la superposition par soustraction* et *la superposition par transparence.*

Superposition par soustraction

Cette technique est relativement simple à comprendre. Etant donné qu'à chaque pixel de l'image est associée une valeur, il paraît intéressant de soustraire la valeur de chaque pixel d'une image de la valeur de pixel de l'image homologue. Cette technique permet aisément de rendre compte des différences entre deux images. L'utilisation d'une telle technique est intéressante dans le cas où il faut évaluer l'état d'un patient sur une longue période de temps. La figure 1.8 représente l'évolution d'une tumeur au cerveau. Afin que le praticien puisse plus aisément évaluer son évolution, la superposition par soustraction a été utilisée. Le résultat est rendu à la figure 1.9.

La figure 1.9 illustre assez bien l'intérêt d'une telle technique. Les parties grisées correspondent à des parties n'ayant pas évolué par rapport à l'image initiale. Les parties en blanc laissent apparaître une lésion croissante, tandis que les parties noires montrent au

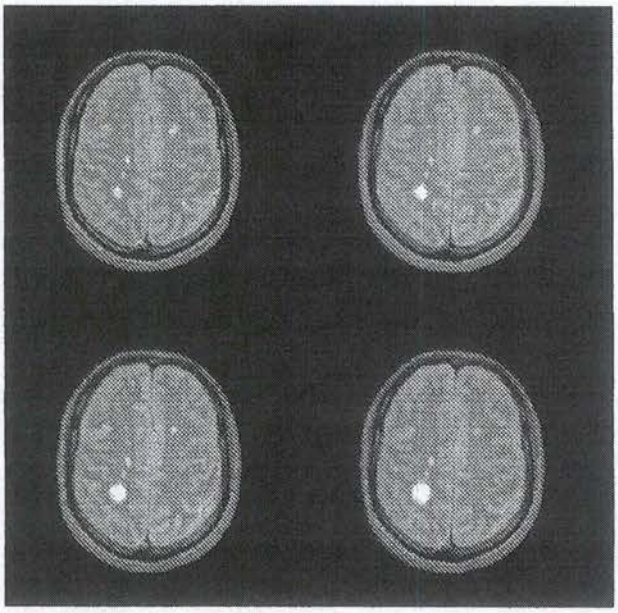


FIG. 1.8 – Evolution d'une tumeur au cerveau. Les images ont été prises à intervalles croissants (IRM) ([Pen96]).

contraire une régression. Le médecin pourra aisément être renseigné de l'évolution de son patient.

Superposition par transparence

La technique se base sur l'idée de donner un effet de transparence à une image et de l'appliquer sur l'autre. Le résultat permettra ainsi de visualiser les éléments des deux images. Le résultat s'obtient en effectuant une combinaison linéaire[10] de chaque valeur associée à chaque point de l'image. Soit *ImgRes*, l'image résultant de la superposition, img_1 et img_2, les deux images. Formellement, le résultat s'exprime par la relation

$$imgRes = \alpha \cdot img_1 + (1 - \alpha) \cdot img_2$$
$$\text{où } \alpha \in]0,1[\subset \Re$$

Le caractère α joue le rôle de paramètre de transparence (ou coefficient d'opacité) appliqué à l'image 1.

[10] On dit que v est une combinaison linéaire de vecteurs v_1, v_2,..., v_r s'il existe les scalaires λ_1, λ_2, ...,λ_r tels que $v = \lambda_1 v_1 + \lambda_2 v_2 + ... + \lambda_r v_r$

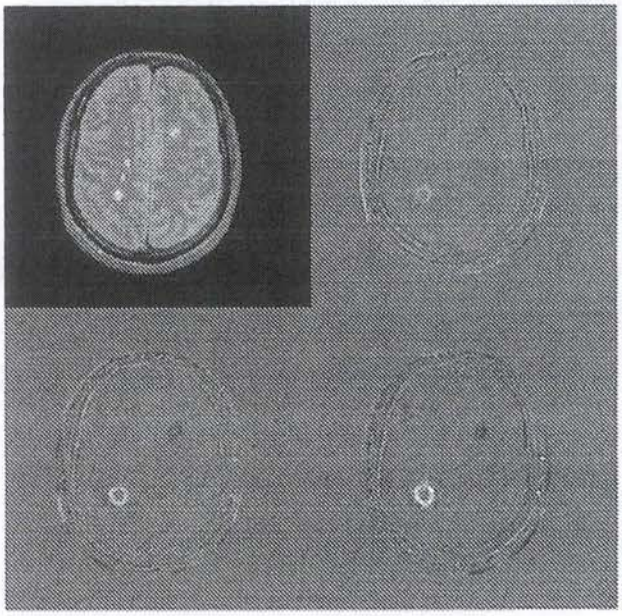


FIG. 1.9 – Superposition par soustraction des images de la figure 1.8 par rapport à la première image. L'intensité des images a été multipliée par cinq pour rendre compte des informations pertinentes ([Pen96]).

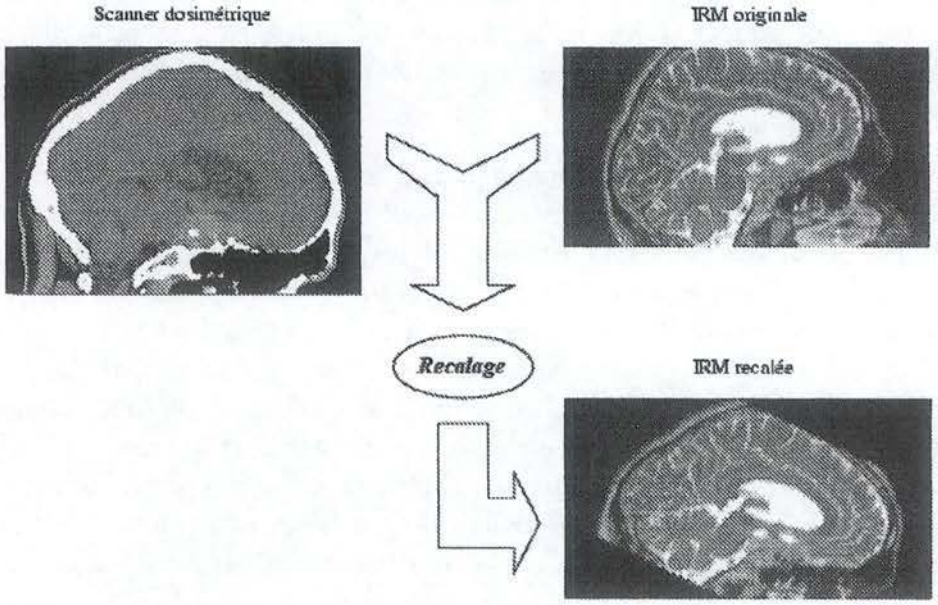


FIG. 1.10 – Etats intiaux et finaux d'une coregistration ([Roc01]).

D'autres techniques existent, permettant par exemple de superposer les images avec des échelles de couleur différentes. Néanmoins, nous n'entrerons volontairement pas dans les détails, la présentation des résultats n'étant pas le thème central de ce mémoire.

Pour conclure cette section, nous dirons que si les techniques d'affichage en parallèle ne dénaturent pas l'information visuelle que procurent les images, elles demandent de la part du praticien un effort pour faire correspondre les points d'une image avec ceux de l'autre. L'affichage par superposition permet par contre d'éviter ce genre d'effort, et la présentation en elle-même peut devenir productrice d'information. Néanmoins, cette technique présente l'inconvénient de modifier le contenu visuel des images. Aussi, allier les deux techniques pourrait être adéquat dans certaines situations.

1.4.3 Composantes intuitives de processus de coregistration

Intéressons-nous à présent au processus de coregistration en lui-même. Cette section tentera, au travers d'un exemple, de faire sentir au lecteur les composantes intuitives de tout algorithme de coregistration. Nous aborderons la notion d'*espace des caractéristiques*, de *transformation* et de *fonctionnelle d'appariement.*

La figure 1.10 illustre les points de départ et d'arrivée d'un recalage. Celui-ci se base initialement sur deux images. L'image de gauche est l'image-maître, celle de droite étant l'image-esclave, ou encore image à recaler. L'image résultante est donc une modification de l'image de départ.

Nous allons décomposer le processus mental d'un individu qui serait amené à devoir apparier manuellement les deux images initiales. Il essaierait certainement dans un premier temps de trouver des *points de repères* sur les deux images, tels que des intensités élevées ou encore des contours significatifs. Une fois son attention focalisée sur ceux-ci, il va essayer de *déformer* l'image à recaler de telle sorte que celle-ci corresponde au mieux avec l'image-maître. Cette déformation s'exprimera peut-être par des translations, des rotations, des étirements ou peut-être même des torsions. Une fois l'image à recaler modifiée, le praticien se penchera sur la question de savoir si la transformation qu'il a effectué s'adapte bien à la première image, autrement dit, s'il y a *concordance* entre les deux images. Dans le cas contraire, il essaiera d'ajuster un peu mieux l'image à recaler. Une fois qu'il s'estimera heureux du résultat, il arrêtera ses modifications.

Au travers de cet exemple, nous avons pu mettre en évidence trois des quatre composantes de tout algorithme de coregistration. Pour les reprendre, nous avons

- **L'extraction de caractéristiques**, qui est l'étape de sélection des points remarquables des deux images
- **La transformation**, qui regroupe l'ensemble des modifications à effectuer sur l'image à recaler
- **La fonctionnelle d'appariement**, qui permet de mesurer la qualité de la concordance entre les deux images

Enfin, la dernière composante, plus difficilement perceptible intuitivement, est **la méthode d'optimisation** qui pourrait être comparée à la capacité de l'être humain à se rapprocher, pas à pas, de la transformation idéale, plutôt que de tenter toutes les transformations possibles de l'image et de retenir celle qui "marche" le mieux.

Cette section achève le chapitre consacré à la mise en contexte et à la mise en place des notions de base. Le chapitre suivant précisera plus en détail les quatre composantes que nous venons de voir. Ces composantes seront un élément clé de l'architecture de la plateforme dédiée à la coregistration d'images médicales. Cette architecture fera l'objet d'une étude détaillée au chapitre cinq.

Chapitre 2

Caractéristiques des algorithmes de coregistration

Introduction

Durant la dernière section du chapitre précédent, nous avons essayé de faire sentir au lecteur les différentes composantes de toute méthode de coregistration. Nous formaliserons, au cours des quatre premières sections de ce chapitre, ces composantes. Nous nous devons d'apporter plus de précisions à ces concepts car l'architecture de la plateforme de coregistration, qui sera explicitée au chapitre suivant, se base très largement sur cette découpe.

La première section abordera la notion d'**espace des caractéristiques**. La deuxième section s'étendra sur le concept de **transformation**. La troisième s'attardera quant à elle sur le concept de **fonctionnelle d'appariement** et, enfin, la dernière explicitera la notion de **méthode d'optimisation**.

Notons que cette partie a déjà fait l'objet d'une étude approfondie dans le mémoire de Stéphane Seynave ([Sey02]). Le lecteur intéressé y trouvera de plus amples descriptions.

2.1 Espace des caractéristiques

Lorsque nous avons essayé de faire sentir d'une manière intuitive les divers composantes d'un processus de coregistration, nous avons dans un premier temps fait ressortir le fait de devoir trouver des points de repère sur l'image-maître et de trouver leurs équi-

valents sur l'image à recaler. L'espace des caractéristiques est un concept assez proche de cette idée. Il consiste en l'ensemble des informations extraites des deux images à co-registrer, utilisé pour caractériser celles-ci et sur lequel va se baser le recalage. Ce sont ces informations qui serviront au calcul de la transformation. Bien entendu, il est évident que cet espace doit être représentatif des points communs entre les images considérées, sous peine d'obtenir des résultats de recalage parfois décevants.

Il est possible de classifier les caractéristiques selon qu'elles reposent uniquement sur le contenu des images médicales ou qu'elles se basent sur des objets extérieurs au patient (cfr. figure 2.1). On dénommera par **caractéristiques intrinsèques au patient** la première catégorie de caractéristiques et par **caractéristiques extrinsèques au patient** la seconde. A la section suivante, nous ne détaillerons que peu les principes liés à la seconde catégorie. En revanche, les caractéristiques intrinsèques au patient feront l'objet d'une description plus détaillée.

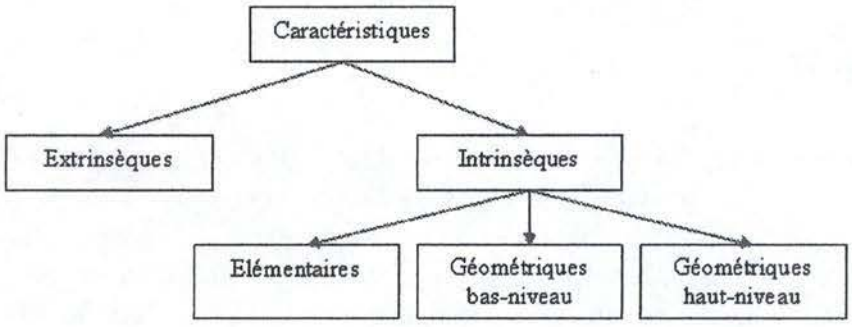


FIG. 2.1 – classification des caractéristiques

2.1.1 Les caractéristiques extrinsèques au patient

En ce qui concerne les caractéristiques extrinsèques au patient, nous ne les détaillerons que peu, car l'architecture que nous décrirons par la suite ne s'inspire pas de ce type de caractéristiques. Nous dirons simplement que ces caractéristiques nécessitent l'emploi de marqueurs externes (masques, repères collés à la peau, ...), qui se doivent d'être visualisables et reconnaissables sur l'image. Il faut bien entendu que les patients portent ces marqueurs non seulement durant toute la durée de l'examen mais également entre les différents examens, afin que les repères que ces marqueurs constituent ne soient en aucun

cas bouleversés, ceci en vue de coregistrer facilement les images. Si l'avantage principal réside dans les performances lors du calcul de la transformation à appliquer à l'image à recaler, cette technique n'est pas exempte de tout inconvénient. En effet, étant donné que les marqueurs doivent être positionnés avant l'examen, cette méthode n'est pas rétrospective. En effet, le positionnement des marqueurs aura été modifié entre des acquisitions effectuées à des moments différents, rendant difficile le recalage. Pour pallier à cela, des techniques d'appariement basées sur des marqueurs intrinsèques au patient existent. Nous allons les aborder au paragraphe suivant.

2.1.2 Les caractéristiques intrinsèques au patient

L'utilisation de caractéristiques intrinsèques (ou anatomiques) pour le recalage est, au contraire des marqueurs extrinsèques, une technique dite rétrospective. Quasiment aucune préparation de l'image n'est alors exigée pour effectuer une coregistration. Les méthodes s'appuient uniquement sur les caractéristiques de l'image. En fonction des primitives utilisées par les méthodes de coregistration, on distingue trois grandes catégories : les **caractéristiques basées sur les voxels, les caractéristiques géométriques de bas niveau** et **les caractéristiques géométriques de haut niveau.**

2.1.2.1 Les caractéristiques basées sur les voxels

Les méthodes basées sur les caractéristiques élémentaires (ou iconiques) reposent sur les valeurs et les positions des voxels uniquement. Dans cette approche, une image est appariée à une autre en effectuant une comparaison des valeurs de gris des pixels entre eux.

Néanmoins, les méthodes de recalage se basant sur ce type de caractéristiques ne sont réellement efficaces que lorsque les images sont issues d'une même modalité ou, plus généralement, lorsque les variations d'intensité au sein des deux images sont plus ou moins les mêmes. Des recalages multi-modalités peuvent néanmoins être envisagés. Lorsque l'on désire recaler une image CT[1] avec une image IRM, on peut effectuer des pré-traitements afin de rendre les images plus semblables. On pourra, par exemple, *mapper* l'image CT de telle façon que l'intensité des os soit diminuée afin de se rapprocher d'une image IRM.

[1] computed tomography

2.1.2.2 Les caractéristiques géométriques de bas niveau

Les caractéristiques géométriques de bas niveau permettent de pallier aux inconvénients discutés au chapitre précédent. Les méthodes exploitant ce type de caractéristiques se basent sur la détection de ce que nous pourrions appeler des caractéristiques de contrôle, telles que des points significatifs, des courbes, des contours ou encore des surfaces. Une fois ces caratéristiques extraites des images, le recalage des images peut être effectué. Celui-ci s'avère beaucoup plus rapide que pour les méthodes basées sur les caractéristiques élémentaires. Le résultat s'avère en outre plus stable.

Cependant, les machines sont généralement peu à l'aise pour détecter ces caractéristiques géométriques. Et le résultat du recalage offrira parfois des résultats complètement faux. Il faudra alors recourir à des méthodes dites semi-automatiques, via lesquelles l'utilisateur guidera la machine dans le choix des repères géométriques.

Le principal inconvénient de telles méthodes est l'absence de sémantique liée aux objets traités. En effet, il est tout à fait possible qu'une transformation soit trouvée de telle sorte que la distance[2] entre les deux objets considérés soit minimale, alors que les objets considérés sont différents. Les méthodes basées sur les caractéristiques de haut niveau évitent ce type de problèmes.

2.1.2.3 Les caractéristiques géométriques de haut niveau

Cette classe de méthodes ajoute une sémantique aux objets considérés. Ces méthodes fonctionnent sur le principe suivant. Dans un premier temps, elles vont extraire des caractéristiques géométriques de bas niveau, lesquelles vont être dans un second temps comparées à une base de donnée contenant des objets étiquetés et standardisés. Les objets se trouvant dans les images sont alors reconnus et reconstruits. Après coup, ces méthodes retirent des images à recaler les éléments non reconnus et inintéressants. Une fois cette étape réalisée, le processus de recalage peut être effectué entre les deux images.

Le point critique de cette classe de méthode réside dans la bonne reconnaissance des différents objets. Cette étape est complexe. En effet, en fonction des circonstances, un objet pourrait être représenté différement. Par exemple, il se pourrait qu'une image SPECT[3] rende une image d'organe fortement différente de celle qui aurait été prise auparavant. D'où, il est nécessaire d'avoir à disposition des bases de données complètes et que l'on puisse facilement mettre à jour.

[2] nous reviendrons sur la notion de distance ente des images à la troisième section de ce chapitre
[3] simple photon emission computed tomography

2.2 Transformation

A travers cette section, nous allons tenter d'identifier les différents types de transformations qui pourront s'appliquer aux images à recaler. Cette distinction portera sur la nature de ces transformations.

2.2.1 Représentation des transformations et système de coordonnées homogènes

On aurait facilement tendance à représenter chaque point (x, y, z) d'une image en trois dimensions par un vecteur colonne $(x, y, z)^t$[4]. Néanmoins, pour des raisons que nous expliquerons plus bas, nous préfèrerons utiliser la représentation en *coordonnées homogènes*. Celle-ci exprime un point d'un espace à n dimensions par un vecteur à $n + 1$ dimensions. Ainsi, le point (x, y, z) sera représenté par un vecteur à quatre dimensions (x', y', z', w) avec la relation

$$\frac{x'}{w} = x, \frac{y'}{w} = y, \frac{z'}{w} = z$$

où w est une constante quelconque différente de 0.

L'introduction de cette $n + 1^{eme}$ composante implique une infinité de vecteurs représentant ce point (x, y, z). En effet, il suffit de multiplier chaque composante du vecteur en coordonnées homogènes par une constante pour avoir un nouveau vecteur relatif à la même coordonnée cartésienne.

On est bien évidemment en droit de se poser la question de l'intérêt d'une telle représentation. La représentation par coordonnées homogènes va permettre de traiter n'importe quelle opération de transformation (qu'il s'agisse d'une translation, d'une rotation etc...) de la même manière, c'est-à-dire par une multiplication de matrice uniquement. En effet, tandis qu'en coordonnées cartésiennes, une rotation nécessite une multiplication par une matrice dite de rotation[5] et une translation nécessite une addition avec un vecteur, les coordonnées homogènes permettront de traiter ces deux transformations par une simple multiplication. La composition de plusieurs transformations se fera donc d'une manière uniforme.

[4] le$_t$ représentant le vecteur transposé

[5] nous en reparlerons plus loin

$$T = \prod_i T_i$$

Plus généralement, toute transformation linéaire tridimensionnelle pourra être représentée par une matrice 4x4 telle que

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = T \times \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \; avec \; T = \begin{pmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{pmatrix}$$

Etant donné la redondance qu'introduit la notation en coordonnées homogènes, il peut exister une infinité de matrices correspondant à une transformation bien précise. On veillera donc à normaliser la matrice de transformation en divisant tous ses élements par le dernier.

2.2.2 Classification des transformations

Nous allons classifier les transformations selon les propriétés géométriques qu'elles conservent. Nous distinguerons alors trois grandes classes : les transformations **rigides**, les transformations **semi-rigides** et les transformations **courbes**. Les sections suivantes expliciteront ces trois catégories.

2.2.2.1 Les transformations rigides

Une transformation est dite rigide si elle conserve la distance entre les points. Elle est simplement composée de translations et de rotations. Ce type de transformation s'applique généralement sur des images représentant des objets relativement stables, telle la tête. Elles sont usuellement utilisées dans les recalages intrasujet, le recalage intersujet requiérant des transformations moins restrictives, la différence de morphologie entre les individus ne pouvant être résolue par des translations et des rotations seulement.

Nous allons illustrer à présent la représentation des différentes transformations rigides et ce, en reprenant la matrice T illustrée ci-dessus.

Translation Pour translater un point (x, y, z) par le vecteur (tx, ty, tz), il suffit d'appliquer la matrice de translation suivante

$$T_t = \begin{pmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation

- Pour pivoter un point autour de l'axe X selon un angle α

$$T_{rx} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Pour pivoter un point autour de l'axe Y selon un angle α

$$T_{ry} = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Pour pivoter un point autour de l'axe Z selon un angle α

$$T_{rz} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Comme nous l'avons dit plus haut, ce type de transformations ne s'applique que dans le cas où la forme des objets est semblable, indépendamment du repère dans lequel on les considère. Néanmoins, ce genre de situation est relativement rare, de par le fait que les organismes sont en perpétuelle évolution et que l'acquisition d'un même organe, tel le coeur, peut présenter des différences assez importantes selon le temps passé entre les deux acquisitions. De même, la coregistration multimodale nécessitera la plupart du temps des transformations moins restricitives.

2.2.2.2 Transformations semi-rigides

Nous aborderons ici deux types de transformations : les transformations affines et les transformations projectives.

Les transformations affines Ce type de transformations possède comme propriété de conserver l'alignement, le parallélisme, les rapports de longueur et de surface. On remarquera que cette définition s'applique également aux transformations rigides qui ne sont, en fait, que des cas particuliers des transformations semi-rigides, dont la matrice de transformation T est de la forme

$$T_a = \begin{pmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Une transformation rigide étant une composition (soit une multiplication dans le cas des coordonnées homogènes) de rotations et de translations, elle s'insère donc dans le cadre de la matrice T_a décrite ci-dessus. Outre les composantes translationnelles et rotationnelles, on y retrouve également des transformations d'échelle et de cisaillement.

transformation d'échelle Ce type de transformation permet le redimensionnement d'un objet. Il peut être représenté par

$$T_{fe} = \begin{pmatrix} fe_x & 0 & 0 & 0 \\ 0 & fe_y & 0 & 0 \\ 0 & 0 & fe_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

où fe_i représente l'étirement appliqué dans la direction i

transormation de cisaillement Ce type de transformation provoque des distortions d'objets en changeant la valeur d'une coordonnée par l'addition d'une fonction linéaire des autres coordonnées.

$$T_c = \begin{pmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ b & c & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Nous terminerons ce paragraphe consacré aux transformations affines en soulignant que celles-ci sont utilisées dans le cadre de recalages multi-modalités. Néanmoins, pour coregistrer des images provenant de modalités spécifiques, ce type de transformation n'est pas encore suffisant. On se devra d'utiliser des transformations de type projectives.

Les transformations projectives Au contraire des transformations affines, ces transformations ne conservent généralement pas le parallélisme. Néanmoins, les lignes droites le sont. Elles sont utilisées en général pour coregistrer des images issues d'une caméra, afin de rendre compte des effets de perspective de l'objet visé. Ce type de transformation étend encore les transformations vues jusqu'à présent.

Une telle transformation est de la forme

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \frac{x_h}{w} \\ \frac{y_h}{w} \\ \frac{z_h}{w} \end{pmatrix} \; avec \; \begin{pmatrix} x_h \\ y_h \\ z_h \\ w \end{pmatrix} = \begin{pmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Ce genre de transformations est principalement utilisé pour coregistrer des images projectives (issues par exemple de radiologie à rayons x) à des images tomographiques en 3 dimensions.

2.2.2.3 Les transformations élastiques

Ces transformations regroupent l'ensemble des transformations qui ne sont ni des transformations rigides, ni des transformations semi-rigides. Nous n'entrerons pas dans les détails de cette section, car ce type de transformations n'a pas fait l'objet d'une modélisation au sein de la plateforme de coregistration. Nous dirons simplement qu'il existe plusieurs sortes de transformations élastiques, telles que les transformations courbes et les transformations fluides. Le principe des premières est de faire correspondre une droite avec une courbe. Généralement, des transformations de type polynomiales seront

appliquées à l'image. Dans le cas d'une transformation fluide, les images sont considérées commes des fluides se déformant en s'écoulant.

Ce type de transformations est utilisé lors de coregistration d'images provenant de patients différents ou lors de comparaison avec des atlas.

Nous avons abordé jusqu'à présent l'espace des caractéristiques, définissant l'ensemble des objets qui seront pris en compte durant le processus de coregistration. Nous avons ensuite défini ce qu'était une transformation. Il nous reste deux composantes de tout processus de coregistration à analyser : la fonctionnelle d'appariement et la méthode d'optimisation.

2.3 Fonctionnelle d'appariement

Lorsque nous avons essayé de percevoir les différents éléments constitutifs de tout processus de coregistration, nous avons défini intuitivement la fonctionnelle d'appariement comme étant une façon de mesurer la concordance entre l'image modèle et l'image recalée (sur laquelle a été appliquée la transformation).

Pour comprendre le rôle exact de cette composante, il faut savoir que la plupart des algorithmes de coregistration sont de type itératif[6], c'est à dire qu'ils recherchent, au fur et à mesure, les paramètres de la transformation optimale. Pour pouvoir estimer la qualité du recalage selon les paramètres trouvés à une certaine itération, il faut pouvoir quantifier la similitude entre les deux images. Conceptuellement, deux approches peuvent être considérées dans cette recherche de paramètres. On itérera jusqu'au moment où soit la similitude entre les deux images est maximale, soit la différence entre les deux images (ou encore la distance entre les images) est minimale. Cette notion de différence, de similitude, constituera le critère objectif dans cette recherche à la transformation optimale. Ce critère est la fonctionnelle d'appariement.

Nous allons dans cette section, non pas d'énumérer l'ensemble des fonctionnelles d'appariement (en dresser une liste exhaustive serait utopique), mais plutôt de les classifier. Nous distinguerons deux grandes classes : les fonctionnelles basées sur la notion de corrélation et celles basées sur la notion de distance. Nous parlerons également d'un autre type de fonctionnelles, basées sur la théorie de l'information, qui adopte une démarche quelque peu différente.

[6]Il existe néanmoins des algorithmes de type déterministes, où la fonctionnelle ne dirige pas le processus de coregistration mais constitue seulement un moyen de mesurer la qualité du recalage final.

2.3.1 Les fonctionnelles basées sur la notion de corrélation

Si nous désirons effectuer une mesure de la similarité entre deux images, le concept de corrélation s'impose assez facilement à l'esprit. Elle consiste alors à mesurer en général la similitude des images quant à leurs niveaux de gris[7]. En considérant $f(x, y, z)$ la valeur de gris d'une image à la coordonnée (x, y, z), ainsi que son équivalent $g(x, y, z)$ pour une seconde image, la mesure de la corrélation entre les deux images, pour autant que f et g soient centrées, peut être la suivante

$$R^2 = \frac{(\sum_{x,y,z} f(x,y,z) \cdot g(x,y,z))^2}{(\sum_{x,y,z} f^2(x,y,z)) \cdot (\sum_{x,y,z} g^2(x,y,z))}$$

Ainsi, plus la corrélation entre les deux images sera importante, plus les deux images seront semblables. Compte tenu de ce qui a déjà été dit au sujet des modalités et des grandeurs physiques que les images représentent, le coefficient de corrélation tel qu'il est exprimé ci-dessus rendra difficilement compte de la similarité entre des images issues de modalités différentes.

Diverses mesures de similarité ont été proposées dans la littérature pour remédier à ce problème. On citera notamment *le coefficient d'inter-corrélation*. En fonction du volume de recouvrement (c'est-à-dire la partie censée être commune aux deux images et choisie pour mesurer la qualité du recalage) et de la valeur moyenne des images observées, la formule est adaptée. Soit p une image de modalité TEP et m une image de modalité IRM, on a

$$\gamma = \frac{\sum_{x,y,z}[p(x,y,z) - \tilde{p}][m(x,y,z) - \tilde{m}]}{\sqrt{\sum_{x,y,z}[p(x,y,z) - \tilde{p}]^2 \sum_{x,y,z}[m(x,y,z) - \tilde{m}]^2}}$$

où $\tilde{p}$ et $\tilde{m}$ représentent l'intensité moyenne des images observées. Notons qu'il faut une certaine corrélation entre les intensités des voxels d'une modalité à l'autre pour que cette mesure puisse rendre effectivement compte de la similarité des deux images.

[7]Notons qu'il existe également des fonctionnelles d'appariement basées sur la corrélation des caractéristiques géométriques. Par une binarisation de l'image, c'est-à-dire en valorisant les pixels appartenant à une caractéristique géométrique reconnue, la mesure de la similarité entre deux images est alors grandement simplifiée

2.3.2 Les fonctionnelles basées sur la notion de distance

Tandis que les fonctionnelles basées sur la notion de corrélation essaient de maximiser la mesure de similarité entre deux images, la notion de distance tend à minimiser la distance entre certains points significatifs de l'image (caractéristiques géométriques, marqueurs cutanés etc...). On peut alors concevoir d'effectuer une mesure uniquement sur ces points ou objets. Pour ce faire, on introduit la notion de distance euclidienne. En supposant X_i (respectivement Y_i) un point significatif de l'image-maître (respectivement l'image à recaler), T une transformation et N points significatifs, alors la distance D peut s'exprimer

$$D^2 = \sum_{i=1}^{N} (X_i - T(Y_i))^t (X_i - T(Y_i)) \quad avec\ X_i = (x, y, z)^t$$

Cette technique peut également s'appliquer quand l'utilisateur spécifie lui-même sur les deux images les points significatifs. Cette saisie manuelle peut alors induire une certaine imprécision et donc un recalage moins efficace.

2.3.3 Les fonctionnelles basées sur la théorie de l'information

Nous allons expliquer, sans entrer dans les détails de la modélisation mathématique, les principes de cette classe de fonctionnelles.

Pour comprendre le fonctionnement de celle-ci, considérons un instant un recalage d'images issues de la même modalité et du même patient. Si les deux images sont *parfaitement* coregistrées, leurs signaux doivent être identiques. Dans le cas du recalage multimodal, par exemple une IRM et un scanner, on ne peut appliquer ce genre de raisonnement.

Néanmoins, on ne peut pas dire que les images sont totalement indépendantes. En effet, ces images étant issues d'un même patient, elles ont forcément une certaine corrélation. Il serait possible d'imaginer une fonction F qui permettrait de prédire (ou du moins approximer) la valeur d'un voxel de l'image scanner à partir de l'image IRM. A partir de cette fonction F, il serait alors possible d'imaginer la mesure du recalage entre les deux images issues de modalités différentes comme un recalage unimodal, par comparaison de l'image IRM et de l'image $F(scanner)$.

Mais pratiquement, trouver cette fonction F est un problème extrêment complexe,

c'est pourquoi on ne cherchera pas à trouver cette fonction, mais plutôt à quantifier l'information mutuelle entre les deux images. Le principe d'une telle technique sera alors de maximiser l'information mutuelle entre les deux images, car on considère que l'information qu'un volume donne à l'autre est maximal lorsque les images sont parfaitement coregistrées.

On peut modéliser ce problème mathématiquement de la façon suivante. Etant donné une image modèle f et une image à recaler g, T la transformation permettant de faire correspondre g avec f, on obtient la transformation optimale en maximisant l'information mutuelle I entre f et $T(g)$. On a donc

$$\hat{T} = \max I(f(x,y,z), T(g(x,y,z)))$$

Nous n'entrerons pas plus dans les détails mathématiques dans ce travail. Pour des informations plus poussées quant à la modélisation et l'implémentation d'algorithmes utilisant ces méthodes, le lecteur se référera à [Sey02].

Pour conclure cette section, soulignons qu'il existe encore d'autres types de fonctionnelles d'appariement. Nous citerons notamment les fonctionnelles basées sur la théorie des modèles déformables et des démons, développées au sein de l'équipe de recherche *Epidaure* de l'*INRIA*.

Finalement, nous avons jusqu'à présent mis en évidence l'espace des caractéristiques, la transformation et la fonctionnelle d'appariement. Il nous faut désormais essayer de mettre en évidence la manière de trouver les paramètres de transformation, telle que l'application de cette transformation sur l'image à recaler optimise la fonctionnelle d'appariement. Cette manière de *balayer* l'espace des paramètres possibles sera abordée à la section suivante.

2.4 Méthode d'optimisation

Nous avons abordé les différents types de transformations possibles et nous avons rendu compte de différentes fonctionnelles d'appréciation de recalage. Il nous faut à présent trouver les paramètres de transformation rendant ce recalage optimal, c'est-à-dire trouver les paramètres tels qu'ils correspondent à un minimum global de la fonctionnelle adoptée. Tel est le but des méthodes d'optimisation.

Si l'espace de recherche est peu important, on peut envisager une recherche exhaustive. Mais c'est rarement le cas. Il faudra donc se munir d'algorithmes d'exploration des solutions sur l'espace de recherche. Ceux-ci définissent la manière dont les fonctionnelles atteignent leur coût minimal. Cependant, ces algorithmes ne peuvent généralement garantir que l'on trouve la solution optimale et ce, pour des raisons de rapidité de convergence. Il faudra bien souvent trouver un compromis entre une recherche globale, mais extrêmement gourmande en calculs, et le risque de trouver un minimum de la fonction qui soit local et non pas global. En effet, plusieurs minima locaux peuvent exister. Selon les méthodes d'optimisation employées, la solution tendra plus ou moins facilement vers le bon argument minimum.

Au travers de cette section, nous ne prétendons pas, encore une fois, établir une liste exhaustive des méthodes d'optimisation existantes. Nous dresserons tout au plus les différentes familles de ces stratégies de recherche. Nous citerons les procédures exhaustives, itératives, stochastiques, structurelles, heuristiques, multi-résolution et hybrides. Nous n'en donnerons qu'une brève définition, fort peu complète pour une compréhension totale du fonctionnement des différentes méthodes citées, et nous invitons encore une fois le lecteur intéressé à parcourir [Sey02] pour plus de détails.

2.4.1 Les procédures (quasi-)exhaustives

Comme expliqué ci-dessus, les méthodes (quasi[8]-) exhaustives sont très peu utilisées car l'espace de recherche est bien souvent trop important. Néanmoins, ce type de méthodes peut être envisagé lorsque le nombre de points caractéristiques à faire correspondre est peu important. Cela peut s'appliquer à une sélection interactive de points sur des images. Lorsque ceux-ci ont été choisis et que leur nombre n'est pas trop important, un tel algorithme peut être envisagé.

2.4.2 Les procédures itératives

Les procédures itératives regroupent un ensemble de méthodes nécessitant ou non l'estimation des dérivées partielles de la fonctionnelle. On citera par exemple la méthode de descente de gradient ou Newton-Raphson. Elles requièrent une fonctionnelle quasi-convexe autour de la solution optimale.

Pour illustrer brièvement cette famille, nous revenons sur la notion de gradient. Le gradient d'une fonction donne localement la direction dans laquelle la fonction décroît

[8]Lorsque le domaine de recherche est limité à des transformations "acceptables"

le plus rapidement. C'est sur ce principe que la méthode dite de descente de gradient se base. Elle permet de construire une suite d'approximations de la solution de manière itérative.

Néanmoins, à moins que le positionnement initial des images soit relativement proche de la solution optimale, les méthodes basées sur le gradient risquent de s'arrêter à un minimum local et non global. Des variantes de cette méthode ont vu le jour, telle que la descente avec inertie, qui permet, lorsqu'on arrive à un minimum, de continuer la recherche en remontant la pente. Une paramétrisation de cette inertie permet d'adapter au mieux la recherche. En effet, une forte inertie est intéressante en début de descente, car tout minimum trouvé serait certainement local. Par contre, une inertie faible à la fin de la descente est conseillée, car on se situera surement près d'un minimum global. Notons enfin que pour une fonctionnelle très irrégulière, ce type de méthode d'optimisation est mal adapté.

2.4.3 Les procédures stochastiques ou statistiques

Ces méthodes sont généralement plus compliquées à mettre en oeuvre que les techniques précédentes. Elles recouvrent des techniques variées telles que le recuit simulé ou les algorithmes génétiques. L'avantage de ces méthodes est de fournir un optimum global ou au moins une solution proche d'un optimum global au contraire des méthodes basées sur le gradient.

Le recuit simulé consiste, en effectuant une exploration aléatoire de l'espace d'état (dans notre cas les paramètres de transformation), à favoriser les descentes sans interdire tout à fait les remontées. Un paramètre T dit de température cautionnera l'acceptation (probablilité) de la remontée de la fonctionnelle. Cette température diminuera au fur et à mesure que l'on se rapprochera du minimum global.

Les algorithmes génétiques quant à eux sont inspirés d'observations concernant les mécanismes de sélection naturelle. Le principe global est issu de ces derniers. Une population de chromosomes (représentant les paramètres du problème) initiale est élaborée. Des opérations de sélection et de reproduction au sein de la population sont effectuées de manière à ne conserver que les individus qui présentent de bonnes aptitudes au problème posé. Les chromosomes sont ensuite regroupés par paires et un certain nombre de ces paires (probablité p_c) subira une recombinaison (échange de données entre chromosomes). Une autre part (probabilité p_m) de la population pourra ensuite faire l'objet d'une mutation (modification du chromosome). Cette population viendra finalement remplacer la population initiale et le cycle peut recommencer. Le cycle se termine lorsque la condition d'arrêt est atteinte.

Notons que ces deux méthodes ont ceci en commun qu'ils sont en quelque sorte des filtres du hasard. Leur principale différence réside dans le fait que les algorithmes génétiques traitent des populations d'individus, tandis que le recuit simulé manipule un seul paramètre à la fois.

2.4.4 Les procédures structurelles

Sous cette classe, on rangera les procédures à base d'arbre ou de graphes, ou encore de programmation dynamique. Ces approches consistent à traiter le problème de manière globale. Bien qu'assez coûteuses en temps de calcul, elles assurent une convergence vers le minimum global de la fonctionnelle de similarité.

2.4.5 Les procédures heuristiques

On retrouvera sous cette nomenclature les méthodes qui sont par nature difficiles à classifier. On y retrouvera, par exemple, les méthodes de hachage géométrique ou de relaxation.

2.4.6 Les procédures multi-résolution

Les méthodes que nous venons d'aborder peuvent s'inscrire dans un schéma plus global, dit multi-résolution. Le principe repose sur des recalages successifs de versions d'images correspondant à des niveaux cohérents de résolution, en allant de résolutions grossières jusqu'à des résolutions plus fines. Chaque niveau se base sur les résultats obtenus par l'étage inférieur. Le processus s'arrête lorsque la résolution initiale des images est atteinte.

2.4.7 Les procédures hybrides

Pratiquement, les traitements en imagerie ne se contentent pas d'utiliser une seule méthode d'optimisation. Ainsi, des méthodes dites hybrides consistent à diviser le problème en sous-problèmes plus ou moins indépendants, dont chacun de ceux-ci peut être résolu selon des méthodes différentes. De même, il ne sera pas rare de voir des méthodes d'optimisation qui ne seront en fait que des successions de plusieurs, par exemple, effectuer dans un premier temps un algorithme rapide et moins précis pour ensuite passer la

main à un second, plus minutieux.

Conclusion

La découpe que nous venons de proposer nous a permis de normaliser tout fonctionnement d'algorithme de coregistration. Nous avons souligné la nécessité de dégager un **espace de caractéristiques**, représentant un ensemble d'objets sur lesquels va s'appuyer le recalage. Nous avons également mis en évidence la notion de **transformation** qui formalise les modifications que l'on peut appliquer sur une image. Nous avons aussi expliqué ce qu'est une **fonctionnelle d'appariement**, en distinguant les différentes manières de quantifier le degré de similitude de deux images. Enfin, la **méthode d'optimisation** s'est imposée comme étant la façon de balayer l'univers des paramètres de transformations possibles.

Avec ce chapitre, nous achevons les notions théoriques de ce mémoire. Nous allons à présent pouvoir nous attacher à la description de l'architecture de la plate-forme de coregistration mise au point lors de mon stage. Dans un premier temps, nous décrirons les technologies et outils utilisés pour cette réalisation. Le chapitre suivant en fera l'objet.

Chapitre 3

Matériels et outils

Au travers de ce chapitre, nous décrirons les moyens utilisés lors de l'élaboration de la plateforme de coregistration lors de mon stage aux Cliniques Universitaire UCL Mont-Godinne. Nous aborderons d'une part les technologies de langage mis en oeuvre et d'autre part les packages et formats de fichiers utilisés.

3.1 Une modélisation orientée objet

La démarche classique utilisée par les algorithmes de coregistration, comme AIR, se base sur une approche dite *procédurale* ou structurée, fondée sur des langages tels que le C. Chaque fonction ou procédure est associée à un traitement particulier qui peut être décomposé en sous-traitements jusqu'à obtenir des fonctions basiques.

La programmation orientée objet se base quant à elle sur la manipulation d'objets, c'est-à-dire des ensembles groupés de variables et de méthodes associées à des entités intégrant ces variables et méthodes.

Nous avons, dans le chapitre précédent, décrit les quatre caractéristiques communes à tout algorithme de coregistration. Cette formalisation va nous permettre de considérer l'approche de la coregistration non plus en terme procédural, mais bien sous la forme d'un modèle descriptif assez générique pour englober les différentes notions abordées.

3.1.1 Design Pattern

L'écriture de code orienté objet réutilisable n'est pas chose aisée. En effet, le design doit être assez spécifique au projet, tout en étant assez général que pour être réutilisable dans un futur. Quand un programmeur trouve une bonne solution à un problème, il ne va pas réinventer la roue à chaque nouveau problème. Il va réutiliser encore et encore les solutions qu'il a trouvé antérieurement, si celles-ci correspondent au problème. Ainsi, on rencontrera souvent dans les systèmes orientés objets ce que nous allons appeler des **patterns**, c'est-à-dire des motifs d'implémentation récurrents. Ils aident les concepteurs dans leur implémentation, ils ne doivent plus repartir de zéro.

On pourra donc définir un Design Pattern comme étant la description d'un problème classique, revenant fréquement, et décrivant le noyau de la solution, de telle manière qu'on puisse réutiliser cette solution encore et encore. On considère généralement qu'un Design Pattern possède quatre éléments essentiels :

- **le nom du pattern** permet de résumer en un mot le problème concerné, la solution associée et les conséquences qui en découlent. Cette nomination est bien entendu intéressante dans un contexte de dialogue ou de documentation.
- **le problème** décrit quand appliquer le pattern. Il explique le problème et son contexte. Il peut parfois citer les conditions à remplir pour que le pattern soit applicable.
- **la solution** décrit quant à elle les éléments nécessaires à la mise en oeuvre du pattern, leurs relations, leurs responsablilités et collaborations. Elle est suffisamment générale pour être appliquée dans plusieurs situations différentes. Elle fournit en quelque sorte une description abstraite d'un problème de design.
- **les conséquences** sont les résultats de l'application du pattern. Elles permettent d'évaluer les résultats de l'application du pattern. Dans une optique de sélection d'un pattern, ce facteur est déterminant.

Ces éléments permettent de caractériser chacun des patterns. Nous n'allons bien évidemment pas détailler tous les types de design pattern existants, nous pouvons tout au plus les classifier. Cette classification peut s'opérer selon un double axe.

Le premier, que l'on pourrait nommer le **but**, reflète ce que fait le pattern. On en distingue trois grands types, à savoir les patterns **créateurs**, les patterns **structurels** et les pattern **comportementaux**. Les premiers concernent le processus de création des objets, les seconds traitent de la composition des classes ou des objets et les troisièmes caractérisent la manière selon laquelle les classes ou les objets interagissent et distribuent leurs responsabilités.

La **portée** constitue le second axe et spécifie si le pattern s'applique principalement aux classes ou aux objets. Les patterns s'appliquant aux classes s'occupent des relations entre les classes et leurs sous-classes. Ces relations sont établies au travers du mécanisme d'héritage, et sont fixées à la compilation. Par contre, les patterns d'objets traitent des relations entre les objets. Celles-ci peuvent être changées à l'exécution et sont donc plus dynamiques.

Nous allons à présent aborder un Design Pattern que nous utiliserons dans notre plateforme. Il s'agit du Design Pattern Stratégie.

3.1.2 Le Design Pattern Stratégie

Le Design Pattern Stratégie est un pattern comportemental, c'est-à-dire qu'il traite non seulement de la description d'objets ou de classes, mais surtout de la communication et des responsabilités entre celles-ci.

Le Design Pattern Stratégie définit une famille d'algorithmes, encapsule ceux-ci dans des objets et les rend interchangeables. Ils peuvent ainsi évoluer sans que les clients ne s'en aperçoivent.

Ce type de Pattern est d'application lorsque

- plusieurs classes apparentées ne diffèrent que par leur comportement. Dans ce cas, ce pattern permet d'appareiller une classe avec un comportement parmi plusieurs autres.
- on a besoin de plusieurs variantes d'un algorithme. On pourrait par exemple considérer des algorithmes effectuant la même tâche mais avec des degrés de précision différents, et plus ou moins gourmands en ressources.
- un algorithme utilise des données que les clients n'ont pas à connaître. Ceux-ci ne devront ainsi pas manipuler les données inhérentes à chaque algorithme.
- une classe définit de nombreux comportements, qui figurent dans ses opérations sous la forme de déclarations conditionnelles multiples. Ces conditionnelles peuvent alors être substituées par des stratégies.

Le schéma 3.1 offre une illustration de ce type d'architecture.

Ce modèle est composé de trois participants. On retrouve

- **Strategy** qui définit une interface, c'est-à-dire un ensemble de méthodes que chaque algorithme supporté s'engage à respecter
- **ConcretStrategy** qui est l'implémentation d'un algorithme respectant l'interface

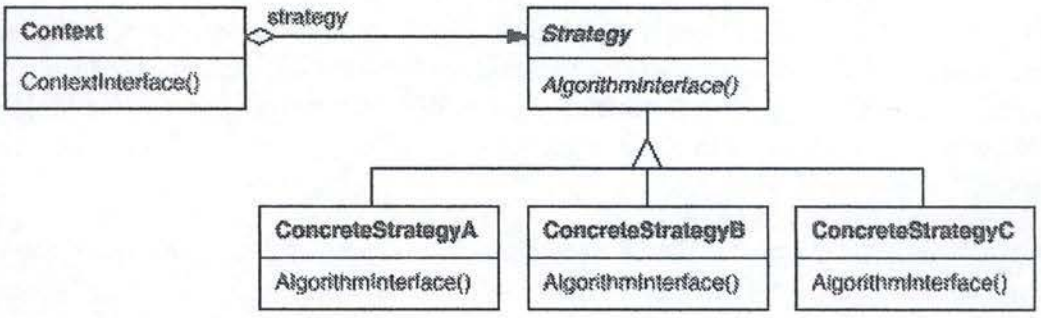


FIG. 3.1 – Design Pattern Stratégie ([GHJV95])

définit par *Strategy*
- **Context**, configuré avec un objet de type *ConcretStrategy* et pouvant définir une interface permettant à un objet *Strategy* d'accéder à ses données.

Nous terminerons la présentation de ce pattern en disant que s'il autorise la définition de familles d'algorithmes, en permettant d'effectuer des choix de manière dynamique et adaptée au contexte, la multiplicité de ces algorithmes engendre une certaine difficulté quant à la présentation des paramètres issus du contexte, qui sont transmis de manière identique à tous les algorithmes concrets[1].

3.2 Java

Nous avons opté pour Java comme langage principal pour le développement de la plateforme. Outre le fait qu'il soit parmi les langages les plus populaires aujourd'hui, son avantage principal réside principalement dans sa portabilité. En effet, l'exécution d'un programme Java est assuré par une machine virtuelle (JVM - Java Virtual Machine), implémentée sous de très nombreux environnements, tels Windows ou Linux. Ainsi, un même code Java pourra être théoriquement exécutable sous n'importe quel environnement, au contraire de programmes écrits en C, qui nécessitent généralement une réécriture.

Néanmoins, l'exécution nécessitera de la part de l'utilisateur une connaissance moyenne de son système d'exploitation, car il devra se montrer capable d'initialiser correctement

[1] Ce problème s'est posé lors de la réalisation de la plateforme de coregistration. Nous en reparlerons dans la description de l'architecture.

les paramètres du programme nécessaires à son bon fonctionnement[2].

3.3 JNI : Java Native Interface

La portabilité de Java et l'existence de sa machine virtuelle induit indubitablement une baisse de performances par rapport à des langages plus dépendants de leur environnement, tel le C. En effet, le processus de compilation d'un programme écrit en Java transforme celui-ci dans un *byte code* indépendant de la plateforme, pour être exécuté par la JVM. Ce *byte code* est alors chargé dans la JVM. Seulement, la machine virtuelle rajoute un niveau entre le programme et le processeur de l'ordinateur sur lequel tourne le programme. C'est cette couche de translation qui est à l'origine de cette perte de performance.

Dans le cadre de processus couteux, tel le calcul de la transformation optimale à appliquer sur une image à recaler, il est compréhensible que la majorité des algorithmes existants aient été écrits dans des langages plus proches de leur environnement.

Nous avons voulu inclure au sein de la plateforme un package de routines dédié au réalignement d'images médicales écrit en C, **AIR**. Il nous fallait donc un moyen de faire cohabiter les deux langages de programmation au sein d'un même programme.

Les *méthodes natives* offrent une solution à ce problème. Une méthode native est essentiellement une fonction déclarée à l'intérieur du code source Java qui est implémentée en utilisant du code *natif*, c'est-à-dire du code écrit dans un autre langage et compilé sous un environnement spécifique. Comme les méthodes natives sont compilées pour cet environnement spécifique, on peut les utiliser pour surpasser les limitations de performances inhérentes à Java. Encore faut-il pouvoir faire appel à ces méthodes à l'intérieur d'un code Java. Le JNI (Java Native Interface) nous permet d'interfacer les programmes Java et les programmes C entre eux (figure 3.2).

Le JNI offre une interface de programmation de méthodes natives, indépendante de l'implémentation de la JVM[3]. Il est implémenté selon une architecture à trois niveaux entre le code natif et la JVM. La figure 3.3 illustre celle-ci.

[2]En ce qui concerne le programme décrit dans ce mémoire, ces paramètres comprennent l'accès aux classes, au code natif et les options de mémoire. Pour plus d'information à ce sujet, voir la partie consacrée à l'utilisation de la plateforme, en annexes.

[3]Historiquement, il existe principalement trois grands types de JVM, chacun supportant sa propre interface de méthodes natives. Ceux-ci incluent les JVMs dérivées de la JVM originale de Sun, de la JVM de Netscape et de son Plugin Development Kit, et enfin de la JVM de Miscrosoft.

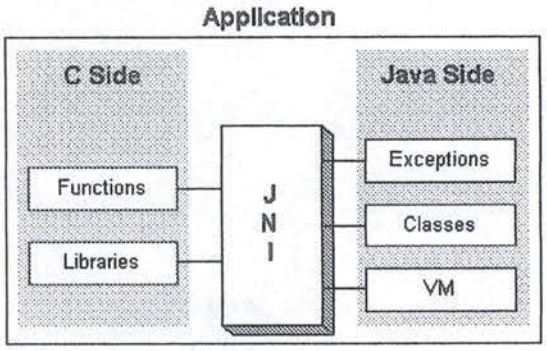


FIG. 3.2 – JNI - Interface entre Java et un code natif ([Webj])

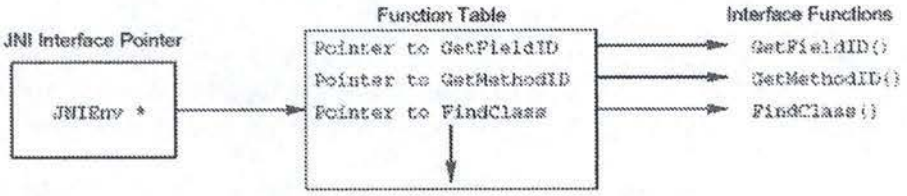


FIG. 3.3 – JNI - Architecture à 3 niveaux ([Des00])

Cette architecture est définie par le *JNI Interface Pointer* qui pointe vers une *Function Table* qui, à chaque entrée, fait correspondre les fonctions JNI ou *Interface Functions*, implémentées au sein de chaque JVM.

Sans entrer dans les détails d'implémentations[4], notons qu'il faut respecter trois tâches lorsque l'on souhaite développer une application supportant le JNI :

- Déclarer les méthodes natives dans les classes Java, grâce au mot reservé *native*
- Ecrire la méthode native elle-même, en utilisant correctement les fonctions prototypes
- Fournir la couche qui permettra au code natif et aux classes Java de communiquer entre-eux, en s'assurant que chacun sache où l'autre se trouve.

Cette technologie a été utilisée afin d'utiliser le package **AIR** au sein de la plateforme. Nous présentons cet ensemble d'outils dédiés à la coregistration dans la section suivante.

[4]Plus d'informations sur les méthodes natives se trouvent en annexes.

3.4 Le package AIR

Conçu par *LONI*, Laboratory of Neuro Imaging, de l'université de Californie, à Los Angeles, *AIR* est un ensemble d'outils dédiés à la coregistration d'images médicales. Il est le premier algorithme à avoir été inclus au sein de la plateforme de coregistration.

Le choix s'est porté vers ce package car ses sources sont en libre accès. De plus, il jouit d'une certaine reconnaissance et fait l'objet de mises à jour régulières.

Pratiquement, l'ensemble du package n'a pas été intégré dans la plateforme, seulement quelques outils ont été utilisés. C'est pourquoi nous allons quelque peu expliciter le contenu du package en abordant les différents types de fichiers utilisés et les grandes classes de programmes. Parmi ces derniers, nous choisirons celui qui permet de calculer la transformation optimale à appliquer sur l'image à recaler. Dans le chapitre suivant, nous aborderons plus en profondeur la manière dont s'effectue la recherche de la transformation via le programme choisi.

3.4.1 Les fichiers

Nous allons donc dans un premier temps décrire les différents types de fichiers utilisés et générés par les différents modules du package. Les fichiers images utilisés par AIR sont au format ANALYZE 7.5. Pratiquement, une image au format ANALYZE est composée de deux fichiers, un premier contenant l'en-tête de l'image (*.hdr*) et un autre contenant les données brutes de l'image (*.img*).

On trouvera également d'autres types de fichiers, générés et utilisés par les différents outils de la bibliothèque. On citera notamment les fichiers contenant la transformation à appliquer sur l'image à recaler. Ils portent l'extension *.air*[5] ou *.warp* en fonction du type de transformation à laquelle elle se rapporte. Ainsi, les transformations rigides ou semi-rigides (respectivement courbes) se verront encodées dans un fichier *.air* (respectivement *.warp*). A côté de ceux-ci, on trouvera encore d'autres types de fichiers, qui ne nous intéresseront pas dans le cadre de cette étude de cas. Nous citerons les fichiers *.ucf*, qui contiennent une liste de coordonnées de points, les fichiers *.init*, qui sont des fichiers d'initialisation, notamment de prépositionnement. Nous n'aborderons pas non plus les fichiers *.warp*, car les types de transformations qu'ils concernent ne seront pas reprises dans la plateforme qui sera décrite dans le chapitre suivant.

[5]Nous reviendrons sur le format de ces fichiers dans l'analyse du processus de coregistration. En effet, ce format permettra de déduire une des quatre composantes abordées au chapitre précédent.

3.4.1.1 Les en-têtes ANALYZE

Les données techniques de l'image médicale sont contenues dans un fichier séparé des données brutes et portant l'extension *.hdr*. On y retrouve les informations nécessaires à la bonne compréhension des données brutes. Sont inclus

- le nombre de bits utilisé pour encoder chaque pixel
- les différentes dimensions de l'image
- la tailles des voxels en mm
- le minimum global de l'image
- le maximum global de l'image
- un entier égal à la taille en byte du fichier d'en-tête[6]

3.4.1.2 les données brutes ANALYZE

Les données brutes de l'image (c'est-à-dire la valeur de chacun des pixels) sont contenues dans un fichier *.img*, de même préfixe que le header correspondant. Ainsi, si nous avons une image *MonImage* au format ANALYZE, nous aurons un fichier *MonImage.hdr* et un autre *MonImage.img*. Par combinaison des données contenues dans l'entête, *AIR* reconnaît quatre types différents de données. Le tableau suivant illustre ces différents types.

dataType	bits/pixel	minimum global	maximum global	value range
0	8	non pertinent	non pertinent	0..255
1	16	$. \geq 0$	$. > 32767$	0..65535
2	16	$. \geq 0$	$0 \leq . \leq 32767$	0..32767
3	16	$. < 0$	$. > -32768$	-32768..32767

Une fois l'en-tête du fichier lue, les modules *AIR* seront capables d'interpréter les données contenues dans le fichier de données brutes, en sachant que l'organisation des voxels se fait d'abord selon l'axe des X puis selon l'axe des Y et enfin selon l'axe des Z.

[6] Ce nombre est destiné à savoir si le fichier a besoin d'être *swappé*. L'architecture des machines influe sur la manière dont sont stockées les données. On distinguera deux façons de stocker l'information. Une architecture dite *little endian* stockera les bytes les moins significatifs à l'adresse la plus basse tandis qu'en *big endian*, ce seront les bytes les plus significatifs qui seront stockés à l'adresse la plus basse.

3.4.2 Le package

AIR est une bibliothèque assez étoffée d'outils dédiés à la coregistration d'images médicales. Il regroupe en son sein diverses catégories d'outils permettant de mener à bien diverses opérations sur ces images, chacune de ces catégories aura une fonction particulière. On citera les catégories qui permettent de

- Créer des fichiers *.air*
- Combiner ou modifier des fichiers *.air*
- Créer de fichiers *.warp*
- Combiner ou modifier des fichiers *.warp*
- Créer ou modifier des fichiers *.hdr*
- Créer des fichiers image
- Créer ou utiliser des fichiers *.ucf*
- Afficher des informations sur les différents fichiers
- Créer des fichiers *.init*

Notre attention se portera sur deux programmes parmi l'ensemble du package. Le premier, nommé *alignlinear* appartient à la première catégorie de programmes et produit à partir de deux images la transformation à appliquer sur l'image à recaler. Le second, *reslice*, permet de créer l'image résultant de la coregistration. La figure 3.4 illustre la marche à suivre pour coregistrer deux images.

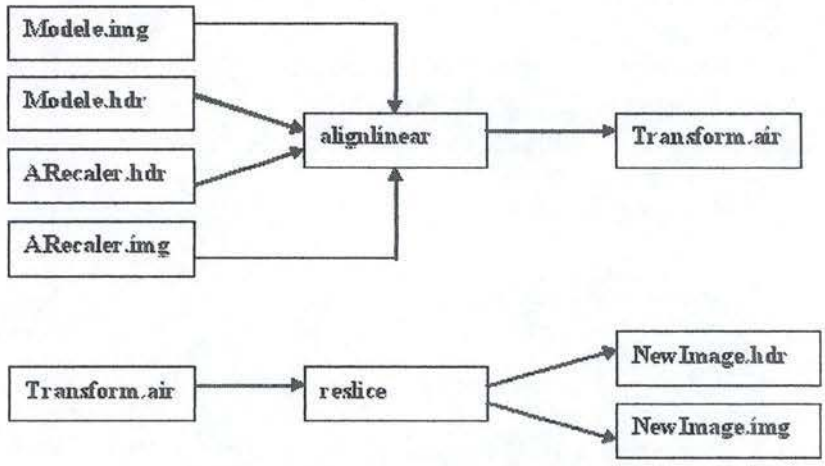


FIG. 3.4 – Fonctionnement d'un recalage avec AIR.

3.5 DICOM

La norme *DICOM* (Digital Imaging and Communication in Medicine) a été créée par l'*ACR* (the American College of Radiology) et la *NEMA* (the National Electrical Manufacturers Association), en collaboration avec d'autres associations d'experts internationaux tels l'*ANSI* (American National Standard Institute) aux USA, le *JRIA* (Japan Research Industries Association) et le *CENTC251* (Commité Européen de Normalisation - Technical Committee 251).

Cette norme fournit non seulement un format spécfique pour les images médicales, mais également des méthodes de connexion, de communication, de transfert et d'identification de données médicales entre les divers équipements d'imagerie médicale. Cette norme tend à devenir le standard de communication entre machines d'imagerie médicale.

3.5.1 Organisation du fichier

Un fichier DICOM comprend, outre les données brutes, des informations administratives. Ces dernières précèdent les données brutes de l'image, qui se situent en fin de fichier. La partie administrative est quant à elle stockée de manière séquentielle au début de fichier. Chaque donnée contient quatre informations.

- **une balise groupe** qui définit le groupe de données dans lequel s'inscrit l'information concernée (par exemple les données relatives au patient)
- **une balise élément** qui spécifie plus précisément l'information (l'âge du patient)
- **une champ de longueur** stipulant la longueur du champ suivant
- **la donnée** qui représente l'information en tant que telle

Pour exploiter ces données, je me suis appuyé sur deux packages, à savoir *zdicom.jar* et *emim.jar*.

3.5.2 zdicom.jar

Ce package a été développé aux Cliniques Universitaires UCL de Mont-Godinne. Il m'a permis d'ouvrir les fichiers DICOM, d'exploiter les différentes données contenues dans l'en-tête du fichier, ainsi que de sauvegarder une image dans le format DICOM.

3.5.3 emim.jar

Ce package a quant à lui été développé par l'équipe *EMIM* (Emission Multimédia des données en provenance de plateaux techniques d'Imagerie Médicale). Il a également permis de récupérer des informations sur les images.

Chapitre 4

Application de la découpe en composantes : AIR

Introduction

Au sein de cette section, nous allons tenter d'indentifier une découpe en composantes du package AIR et plus spécifiquement de son outil de réalignement `alignlinear`. Avant de rentrer au sein du programme alignlinear, nous reviendrons sur le fichier *transform.air* du schéma 3.4. Comme son nom le laisse supposer, ce fichier contient les informations nécessaires à l'application d'une transformation sur une image. Nous allons à présent "disséquer" le fichier pour en ressortir les informations qui nous préoccupent.

4.1 Le fichier de transformation *.air*

Les fichiers de type *.air* contiennent une structure, définie dans le package AIR, appelée *AIR_Air16*. Cette structure contient toutes les informations nécessaires à la réalisation du recalage par *reslice*. On y retrouve

- **le fichier à recaler** : il s'agit du chemin d'accès au fichier image devant être transformé
- **la définition de l'espace** : on y retrouve les dimensions des voxels et de l'image
- **la matrice de transformation** : une description sous la forme d'une matrice 4x4 de la transformation à appliquer
- **des informations additionnelles** : elles expliquent la façon dont le fichier a été

généré.

La première composante est donc très facile à déterminer. La transformation est codée, comme nous l'avons vu dans le deuxième chapitre, sous la forme d'une matrice 4x4. Les composants suivants sont plus difficiles à identifier. Il faut pour cela se plonger dans les entrailles de *alignlinear*.

4.2 le fonctionnement d'alignlinear

alignlinear est un outil de réalignement assez puissant. Il est capable d'effectuer aussi bien des recalages intramodalité que des recalages intermodalité, tout comme il peut effectuer des coregistrations intrasujet ou intersujet. Pour ce faire, il dispose d'une paramétrisation conséquente. On peut par exemple spécifier le type de transformation que l'on désire rechercher, tel que des transformations rigides ou semi-rigides. On peut également spécifier la fonctionnelle d'appariement que l'on souhaite utiliser pour mesurer la similarité des deux images, par exemple l'utilisation de l'écart-type ou la méthode des moindres carrés. Nous ne pouvons pas décrire l'ensemble des méthodes disponibles, c'est pourquoi nous nous concentrerons sur un type de transformation bien spécifique, à savoir les transformations rigides, ainsi que sur la fonctionnelle basée sur l'écart-type.

Pour expliquer le fonctionnement du programme, nous nous mettrons dans le cas où nous voulons coregistrer une image TEP avec une image IRM. Rappelons encore que les images TEP et IRM sont très différentes[1] La coregistration d'une image TEP avec une image IRM est telle que, même si les deux images sont parfaitement coregistrées, il n'existe pas de "simple fonction" qui relie les intensités de voxel de l'une à l'autre. Pour s'en convaincre, on peut considérer le crâne et la matière blanche du cerveau qui produisent les mêmes intensités de voxel au travers d'une IRM, alors qu'elle sont associées à des valeurs tout à fait différentes sur une image TEP. Pour régler ce problème, on aura recours à des prétraitements de l'image (souvent manuels[2]), tels que le retrait de la partie osseuse.

En supposant à présent que les images sont prêtes à être recalées, on peut se pencher maintenant sur le mode de fonctionnement d'*alignlinear*. L'algorithme est basé sur le principe-clé que des intensités de voxels similaires dans l'image IRM correspondent à des tissus de même type (en considérant toujours que les voxels relatifs à la partie de la tête ne correspondant pas au cerveau (les os) sont mis à 0). En considérant également que ce

[1] cfr. section sur les modalités dans le chapitre premier.

[2] il serait intéressant de rendre ces prétraitements complètement automatiques et intégrés dans les routines de coregistration d'*AIR*, ce qui n'est pas le cas pour l'instant

principe de correspondance est valable aussi pour les images TEP, l'idée de l'algorithme est de trouver une transformation T telle que tous les voxels IRM d'une même intensité soient liés à un ensemble de voxels TEP similaires. On peut mesurer cette similarité par le rapport entre l'écart-type et la moyenne.

Plus formellement, l'algorithme va diviser l'image IRM en K partitions P_j selon l'intensité des voxels et va chercher à minimiser le rapport entre l'écart-type et la moyenne des voxels TEP correspondant à P_j, en manipulant la matrice de transformation T. Mathématiquement, la formule clé de l'algorithme est

$$F(TEP, IRM, T) = \frac{1}{N} \sum_{j}^{K} n_j \frac{\sigma_j^T}{a_j^T}$$

où TEP est l'image TEP, IRM l'image IRM, T la matrice de transformation, n_j est le nombre de voxels de l'image IRM appartenant à la classe d'intensité associée à la partition j, $N = \sum_j^K n_j$, a_j^T est la moyenne de toutes les valeurs de voxels de l'image TEP (à laquelle a été appliquée la transformation T) qui correspondent aux voxels IRM de la partition P_j et σ_j^T leur écart-type. La figure 4.1 illustre schématiquement ce vers quoi tend la coregistration.

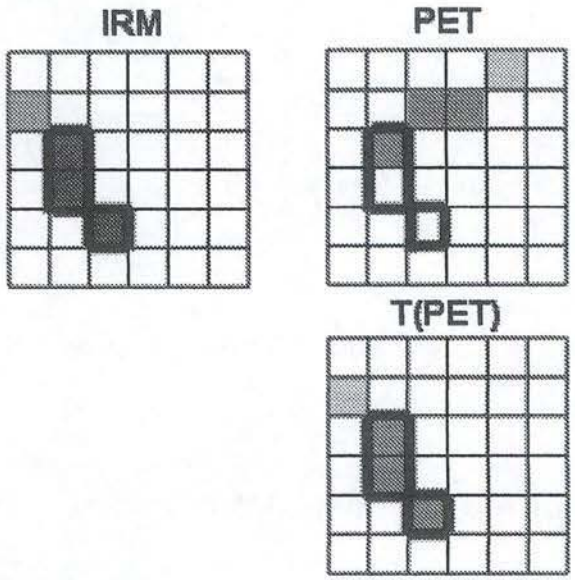


FIG. 4.1 – Coregistration sur les classes d'intensité

Le but de l'algorithme est de trouver une transformation T telle qu'elle minimise le second membre de l'équation ci-dessus. La méthode de minimisation choisie est de

type itérative, chaque étape produisant une nouvelle matrice de transformation T, qui est elle-même utilisée dans l'itération suivante. Lorsque la coregistration est assez bonne, c'est-à-dire lorsque le processus de minimisation arrive à un minimum (global ou local), ou lorsque le nombre d'itérations maximal a été atteint (l'utilisateur peut spécifier celui-ci en paramètre), l'algorithme renverra la transformation trouvée. La figure 4.2 illustre schématiquement le fonctionnement de l'algorithme.

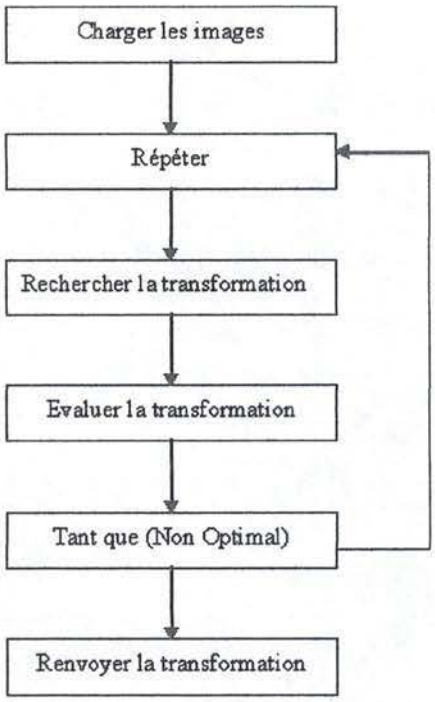


FIG. 4.2 – Schéma représentant le fonctionnement d'alignlinear.

4.2.1 Identification de l'espace des caractéristiques

Comme nous venons de le voir, le programme se base directement sur les valeurs des voxels. On est donc en présence d'un espace de caractéristiques de type élémentaire. Néanmoins, prendre en considération l'ensemble des voxels de l'image pour le recalage est synonyme de perte de temps de calcul. En effet, seule une partie de l'image est intéressante pour la coregistration. C'est pourquoi il est possible de sélectionner un seuil[3] à

[3]threshold dans la littérature

partir duquel les voxels seront considérés. Ce seuil devra être bien choisi, de telle manière que les voxels correspondant au cerveau soient considérés, alors que les voxels se trouvant en dehors ne le soient pas. Ce seuil est un outil à double tranchant car si un seuil haut permet un gain de temps appréciable lors du processus de coregistration, il risque également d'impliquer un recalage parfois imprécis, certains éléments n'ayant pas été pris en compte. Cette valeur devra donc être choisie avec beaucoup de prudence. Le chargement de l'image se fera via la routine[4]

MethodName	Params	Return
AIR_load	Path	Voxel[x_dim][y_dim][z_dim]

où *Path* représente le chemin d'accès à l'image, *Voxel* est le tableau 3D de voxels. Notons que le treshold ne filtre pas les pixels à ce niveau, mais sera considéré lors du calcul de la fonctionnelle.

4.2.2 Identification de la fonctionnelle d'appariement

L'information fournie par l'écart-type est basée sur la notion de corrélation. L'algorithme tente de minimiser la dispersion des valeurs de voxels TEP selon chaque partition P_j. Cette fonction de coût est pratiquement calculée par la routine[5]

MethodName	Params	Return
AIR_uvDerivsN6	TEPVoxel[x_dim][y_dim][z_dim] IRMVoxel[x_dim][y_dim][z_dim] TransformationMatrix Treshold	ecart-type

où *TEPVoxel* est le tableau 3D représentant l'image TEP, *IRMVoxel* est le tableau 3D représentant l'image IRM, *TransformationMatrix* représente l'ensemble des paramètres de la transformation, et *Threshold* spécifie le seuil à partir duquel on considère les voxels. Pratiquement, cette routine calcule pour chaque voxel IRM satisfaisant à la contrainte du treshold son équivalent pour l'image TEP surlaquelle on applique les paramètres de transformation. La valeur de la fonctionnelle sera ensuite calculée.

[4]Les routines nommées ici ont été allégées de certains paramètres, peu important pour une compréhension globale du processus.

[5]idem note précédente

4.2.3 Identification de la méthode d'optimisation

Finalement, la méthode d'optimisation est la fonction globale de recherche de la transformation. Elle peut être considérée comme la boucle du schéma de la figure 4.2. Son rôle est donc d'orienter la recherche de la transformation. Pratiquement, parmi les routines AIR, le calcul de la transformation est réalisé à chaque itération par la fonction

MethodName	Params	Return
AIR_uv3D6	TransformationParams TransformationMatrix	

où *TransformationParams* représente les informations nécessaires à la recherche de la transformation. Nous nous sommes volontairement restreints aux transformations de type rigide. Ce type de transformation implique l'utilisation de six paramètres : la rotation autour de l'axe des x, des y et des z, et les translations selon ces mêmes axes. En fonction de la valeur de la dérivée première et seconde de la fonctionnelle d'appariement, calculée entre l'image modèle et l'image à recaler sur laquelle a été appliquée la transformation représentée par *TransformationMatrix*, ces six paramètres prennent leur valeur. *TransformationMatrix* est alors recalculée pour tenir compte de ces six paramètres. Ainsi peut commencer une nouvelle itération avec évaluation de la qualité du recalage par la fonction *AIR_uvDerivsN6*, calcul des six paramètres, ...

Conclusion

A travers cette section, nous avons découpé l'algorithme en composantes. Néanmoins, l'image que nous en avons donnée est grandement simplifiée. En effet, déterminer l'influence et le rôle de tous les paramètres entrant en compte dans chacune des diverses fonctions exposées est très difficile, l'analyse d'un code source non documenté (ou peu) étant très fastidieuse.

Cette étude de cas nous a permis non seulement de valider la découpe qui a été proposée mais également de "préparer" l'insertion du package au sein de la plateforme.

Au cours du deuxième chapitre, nous avons dégagé un **espace de caractéristiques**, représentant un ensemble d'objets sur lesquels va s'appuyer le recalage. Nous avons également mis en évidence la notion de **transformation** qui formalise les modifications que l'on peut appliquer sur une image. Nous avons aussi expliqué ce qu'est une **fonctionnelle**

d'appariement, en distinguant les différentes manières de quantifier le degré de similitude de deux images. Enfin, la **méthode d'optimisation** s'est imposée comme étant la façon de balayer l'univers des paramètres de transformations possibles.

Après avoir posé toutes ces bases théoriques, nous allons dans le chapitre suivant aborder la plateforme qui a été mise au point lors de mon stage aux Cliniques Universitaires UCL de Mont-Godinne.

Chapitre 5

Architecture et implémentation de la plateforme de coregistration

Introduction

Nous avons lors du premier chapitre mis en place les notions nécessaires à la compréhension du problème qu'est la coregistration d'images médicales. Nous y avons décrit aussi bien les constituants d'une image que le type d'information qu'elle peut rendre. Nous avons terminé ce chapitre par une première approche intuitive des processus de coregistration. Ces processus ont été ensuite détaillés dans le chapitre suivant. Nous avons distingué et expliqué ses diverses composantes. Nous achevions alors les notions théoriques. Au cours du chapitre trois, nous avons abordé les technologies et outils dont nous aurons besoin par la suite. Le package *AIR*, et plus particulièrement son outil dédié à la coregistration, a fait l'objet, au chapitre quatre, d'une découpe analogue à celle proposée au chapitre deux.

Le chapitre que nous abordons est en quelque sorte le point d'orgue de ce mémoire, car il va mettre à contribution tout ce qui a déjà été dit dans les chapitres précédents. Il concerne la description et l'implémentation d'une plateforme orientée objet de coregistration d'images médicales mis au point durant le stage au département de médecine nucléaire des Cliniques Universitaires UCL de Mont-Godinne

Nous allons dans les sections suivantes mettre en évidence la découpe en quatre couches qui a été adoptée. Ces dernières feront l'objet de modélisations approfondies. Nous parlerons des outils développés dans le cadre de ce stage, développés pour supporter les formats d'images utilisés à Mont-Godinne ainsi que ceux nécessités par *AIR*. Nous

évoquerons également les outils auxiliaires de manipulation d'images, également mis au point lors de ce stage.

5.1 Un modèle en quatre couches

Le coeur de l'implémentation de la plateforme se base sur un modèle à quatre niveaux. On trouvera

- au premier niveau, la composante **image médicale** du système. Elle gère la manière dont les images médicales (autant les données brutes ques les données administratives) seront représentées.
- au deuxième niveau, on retrouve la couche relative aux **éléments constitutifs** des algorithmes de coregistration. Elle correspond aux éléments que nous avons mis en évidence dans le chapitre deux.
- au troisième niveau, on retrouvera une couche dite **stratégie**. Cette couche permet d'administrer l'ensemble des stratégies de coregistration possibles. Le Design Pattern Stratégie y sera employé. Nous y définirons ainsi une famille d'algorithmes, encapsulés et interchangeables.
- enfin, la dernière couche est celle de **l'éditeur** dédié à la coregistration, qui gèrera l'environnement graphique, la sélection des stratégies et les opérations classiques telles l'ouverture et la fermeture de fichiers.

5.2 Couche 1 : l'image médicale

Dans le premier chapitre nous avons abordé la notion d'image médicale, en précisant quels étaient ses différents constitutifs. Nous avions déterminé qu'une image médicale était non seulement composée d'une image numérique, c'est-à-dire un ensemble de données brutes, mais également d'une partie administrative, pour pouvoir correctement interpréter les données brutes. On pourra donc scinder la représentation des images médicales en trois parties : **les données administratives**, **les données brutes**, et enfin, **l'image médicale** qui est en fait le lien entre les deux précédentes.

5.2.1 Le contexte médical

La représentation utilisée pour les informations a en fait déjà été partiellement abordée dans le chapitre premier. Nous avions classifié en quatre grandes parties les données

contextuelles à l'image. La représentation en est grandement inspirée. Actuellement, seules les données techniques de l'image, intégrées dans la partie consacrée aux informations générales, ont été codées. La figure 5.1 illustre cette représentation. Notons que les classes ici représentées sans attribut n'ont pas été implémentées dans un premier temps, et ce afin de se concentrer uniquement sur l'aspect coregistration de la plateforme. La présence des divers attributs présents dans la classe `BasicImageInfo` a été dictée par les besoins exprimés par le package *AIR*[1], abordé au chapitre précédent.

Peu de choses sont à dire sur cette partie, si ce n'est qu'elle a principalement un rôle informatif (en dehors des informations techniques de l'image, qui interviennent dans le processus de recalage).

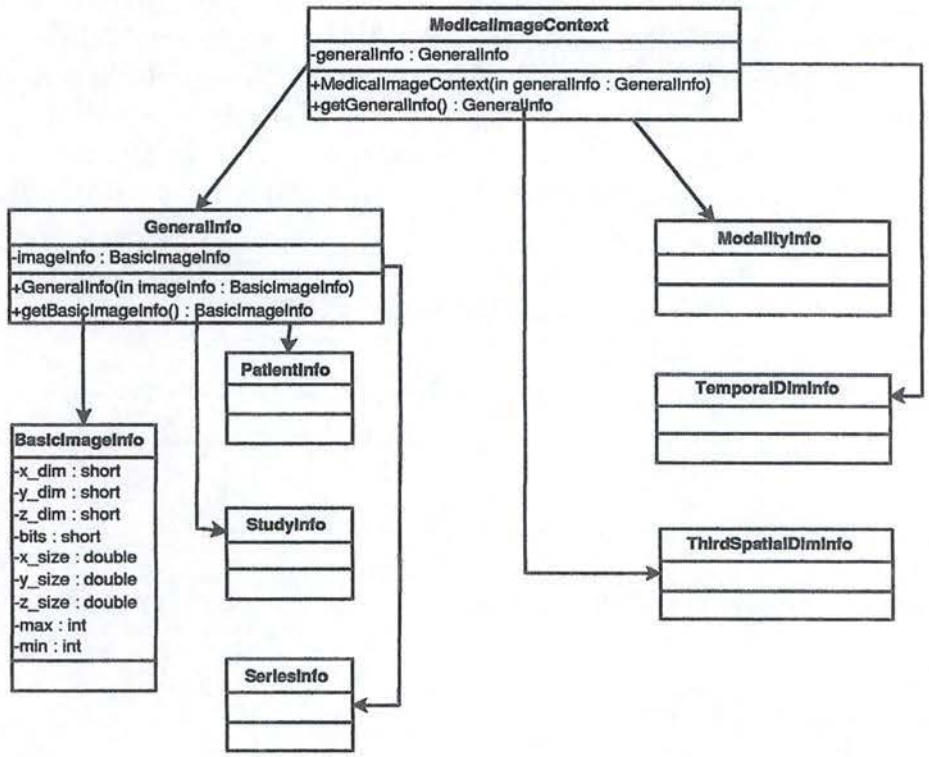


FIG. 5.1 – Représentation du contexte médical

[1]Rappelons encore une fois qu'AIR (plus spéfiquement `alignlinear`) est le premier algorithme de coregistration à avoir été incorporé au sein de la plateforme

5.2.2 Les données brutes

Les données brutes représentent la valeur de chacun des pixels de l'image. Nous avons vu au cours du premier chapitre que ces valeurs pouvaient être codées sur un nombre variable de bits, en fonction des valeurs physiques relatives à la modalité employée. Pour représenter ces valeurs, nous avons opté pour des tableaux dont chacun d'entre-eux représente une tranche du volume en trois dimensions. L'accès aux données de chacune des tranches en est ainsi facilité.

Nous avons donc implémenté des classes pour représenter les différents types de données. Ces classes portent le nom de `RawDataBuffer` suivi du type de donnée à laquelle elle se réfère. Ainsi, la classe qui représente un buffer d'entier (`int` en Java) se nommera `RawDataBufferInt`. Néanmoins, pour une utilisation confortable, chacune de ces classes hérite d'une classe abstraite appelée `RawDataBuffer`. Cette classe possède en outre un attribut (`dataType`) qui permet au programmeur de savoir rapidement quel type de buffer il traite. Pratiquement, il lui suffira de tester la valeur de retour de la méthode `getDataType()`. Le tableau suivant explicite la sémantique de la valeur retournée par cette dernière.

Valeur renvoyée	buffer de
1	byte
2	short
3	unsigned short
4	integer
5	float

Finalement, les relations entres ces différentes classes sont résumées à la figure 5.2

5.2.3 L'image médicale

Enfin, il nous faut représenter la réunion du contexte médical avec le ou les buffers de donnée .Il faut garder à l'esprit que nous devons pouvoir représenter des images non seulement planes, mais également en trois dimensions, voire de dimensions encore supérieures. Pour ce faire, il semble inévitable de devoir utiliser une structure récursive. Il nous faut donc distinguer un cas de base ainsi qu'un cas général.

Il semble clair que l'on peut considérer une image médicale plane comme étant le cas de base. En effet, réduire une image médicale à une seule dimension n'a bien évidemment pas de sens. Si nous désirons par contre passer à une dimension supérieure, c'est-à-dire

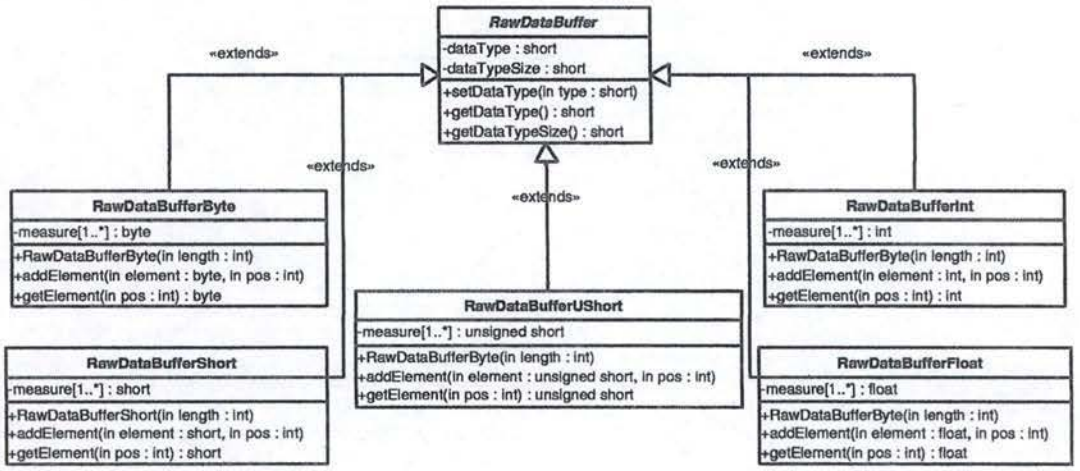


FIG. 5.2 – Structure des buffers de données brutes (*RawDataBuffer*)

représenter des images médicales volumiques, il nous faut alors considérer des ensembles ordonnés d'images planes. De même, si nous désirons passer à une image de dimension encore supérieure, il nous faudra considérer des ensembles ordonnés d'images volumiques.

Nous avons donc comme cas de base l'image plane et comme cas récursif un ensemble d'images (nous parlerons également de pile ou *stack* d'images). Il est certain que les données contextuelles de l'image doivent être communes aussi bien aux images planes qu'aux images volumiques. Par contre, on n'associera de buffers de données brutes qu'aux images bi-dimensionnelles.

Pratiquement, nous aurons donc une classe abstraite `MedicalImage` à laquelle est associée le contexte médical, point commun entre une image plane et un stack d'images.

Une classe concrète est nécessaire pour représenter les images médicales planes. La classe `MedicalImage2D` assurera ce rôle. Outre les attributs dont elle hérite de la classe abstraite, elle possède une référence vers un objet de type `RawDataBuffer`.

`MedicalImageStack` est quant à elle la classe qui assure la représentation d'images de dimension supérieure à 2. De manière générale, une image de dimension n ($n > 2$) se verra composée d'un nombre z d'images de dimension $(n-1)$. `MedicalImageStack` aura donc en son sein une structure ordonnée (tableau) de z objets de type `MedicalImage`, ainsi que les différentes méthodes d'ajout, d'accès et de suppression d'image. La figure 5.3 illustre la structure.

Il est intéressant de noter que, bien que chaque image, considérée dans son entièreté,

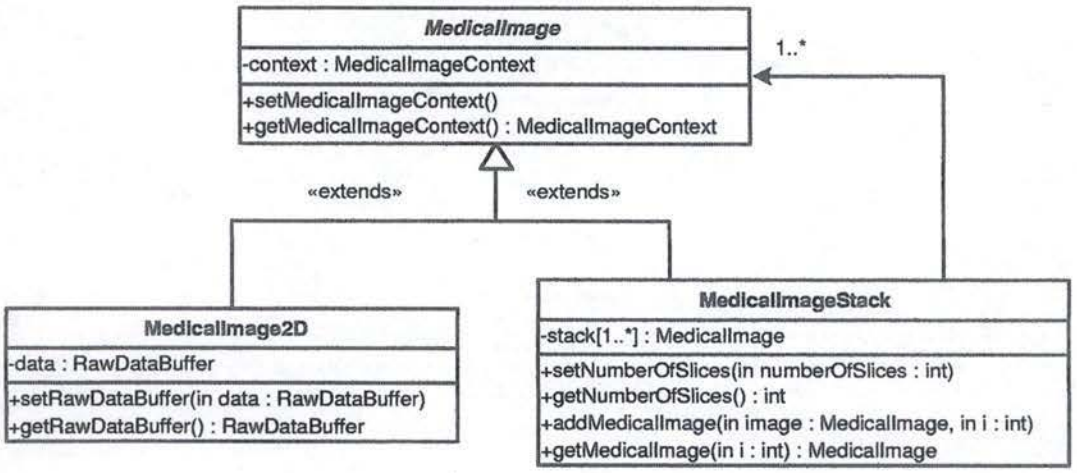


FIG. 5.3 – Structure d'une image médicale

possède des données contextuelles, il est possible d'associer, selon ce schéma, des données contextuelles pour chaque composante de l'image. Ainsi, dans le cas d'une image 3D, il est possible d'associer à chacune des coupes du volume son épaisseur, dans le cas où le système d'acquisition permet de faire varier l'épaisseur des tranches durant l'examen.

5.3 Couche 2 : les éléments constitutifs

Nous avons vu dans le deuxième chapitre qu'il était possible de décomposer un algorithme de coregistration en quatre composantes : l'**espace des caractéristiques**, la **transformation**, la **fonctionnelle d'appariement** et enfin la **méthode d'optimisation**. Nous allons voir comment chacune des parties a pu être modélisée.

5.3.1 L'espace des caractéristiques

Pour rappel, l'espace des caractéristiques consiste en l'ensemble des informations extraites des deux images à coregistrer et sur lequel sera basé le recalage. Les caractéristiques que nous allons modéliser dans cette section sont de type intrinsèque, c'est-à-dire se basant directement sur les données disponibles dans l'image, sans demander une quelconque préparation de la part du patient(positionnement de marqueurs etc...). La figure 5.4 illustre la modélisation que nous allons aborder.

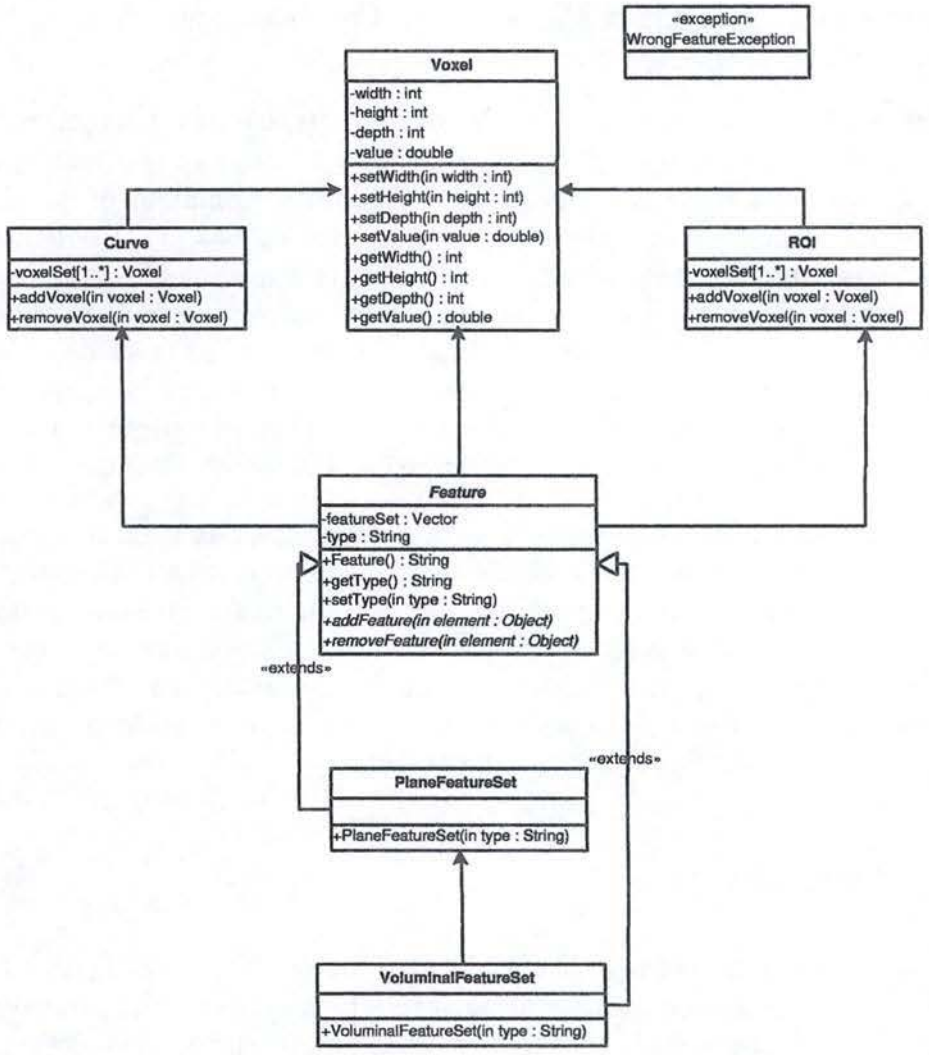


FIG. 5.4 – Modélisation de l'espace des caractéristiques.

Il fallait pouvoir donner un élément de base à toute caractéristique. Le choix s'est rapidement porté vers la notion de voxel. Rappelons qu'un voxel est caractérisé non seulement par une position dans un espace à trois dimensions, mais également par une valeur. La classe `Voxel` représente ce concept. Elle reprend la position et la valeur du voxel , ainsi que les différentes méthodes d'accès aux divers champs.

Il nous fallait pouvoir rendre compte d'un certain nombres de caractéristiques géométriques de base. Des classes ont ainsi été implémentées, ne différant réellement que par leur sémantique. Ainsi on retrouvera une classe représentant une courbe (`Curve`) et

une classe représentant une région d'intérêt(`ROI`). Ces classes regroupent en leur sein un ensemble de `Voxel`.

Finalement, il nous faut pouvoir rendre compte d'ensemble de caractéristiques géométriques. En effet, un recalage ne s'appuiera généralement pas sur un seul point ou sur une seule courbe. Il fallait donc pouvoir représenter des ensembles de caractéristiques géométriques non seulement sur une coupe en deux dimensions, mais également sur des volumes. La classe abstraite `Feature` décrit ce qu'est un ensemble de caractéristiques, en définissant les méthodes abstraites d'ajout et de retrait de caractéristiques. Nous pouvons alors appréhender les ensembles de caractéristiques sur deux niveaux à travers deux classes concrètes étendant `Feature`. Le premier, représenté par la classe `PlaneFeatureSet`, implémente les ensembles de caractéristiques se trouvant dans une même coupe. Le second, `VoluminalFeatureSet`, est dédié à la représentation d'ensembles de `PlaneFeatureSet`.

Néanmoins, nous avons voulu restreindre la manière dont les caractéristiques de base se combinent. En effet, il ne sera pas possible de considérer un ensemble de caractéristiques représenté par une courbe avec une région d'intérêt. Généralement, les algorithmes n'utilisent qu'un type de caractéristiques. C'est pourquoi nous pouvons observer un champ `type` dans la classe `Feature`. Il prendra sa valeur via le constructeur des deux classes héritant de cette dernière. Lorsqu'on essaiera d'insérer un élément différent des précédents, l'exception `WrongFeatureException` sera alors levée.

5.3.2 La transformation

Nous avons dans un chapitre précédent défini les différentes sortes de transformations. Nous avons remarqué que les transformations rigides et semi-rigides pouvaient être représentées par des matrices carrées 4x4. Nous nous basons sur cette représentation pour établir la structure de ce composant. La classe `Transformation` comprendra donc une matrice de `double` (réels double précision), et sera dotée des méthodes d'accès et de modification de chacun des éléments.

`AffineTransformation` représente quant à elle une transformation affine. Ce type de transformation restreint simplement les valeurs possibles des éléments de la dernière ligne (les trois premiers éléments valant 0, le dernier 1). Notons que de par le principe de l'héritage, elle possède toutes les méthodes de `Tranformation`. La figure 5.5 illustre cette modélisation.

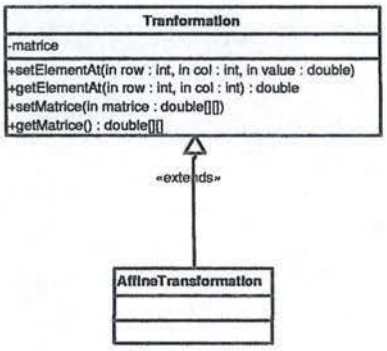


FIG. 5.5 – Représentation des transformations

5.3.3 La fonctionnelle d'appariement

Le troisième composant que nous avions identifié était la fonctionnelle d'appariement, permettant de mesurer le degré de similitude entre deux images. Nous devions dès lors modéliser le concept de fonction appliquée à une image.

Nous sommes partis de l'idée que toute modélisation de fonction, quelle que soit sa nature, doit fournir deux services. Elle doit pourvoir assigner des valeurs aux variables de la fonction qu'elle représente et, selon leur valeur, fournir un résultat. La classe abstraite `PairingFunctional` spécifie ces deux méthodes. On retrouvera donc

- une méthode `execute()` qui permettra de calculer la valeur de la fonction
- une méthode `assign(variable,valeur)` qui spécifiera la valeur d'une variable

Bien entendu, le calcul de la valeur d'une fonction ne peut se faire que si chaque variable apparaissant au sein de la fonction est assignée à une valeur. L'exception `NoAttributedValueE` est présente pour avertir de l'indétermination liée à une de celles-ci. Elle permet au programmeur de le renseigner sur la variable qui n'a pas été initialisée. Cette information est disponible via la méthode `toString()`, réimplémentée pour l'occasion. La figure 5.6 rend compte de cette implémentation.

Toutes les classes implémentées jusqu'à présent n'ont pas été représentées sur cette dernière figure. Le schéma en aurait été encombré sans réellement apporter d'information supplémentaire. Le tableau suivant rend compte des différentes classes implémentées et de ce qu'elles permettent de représenter.

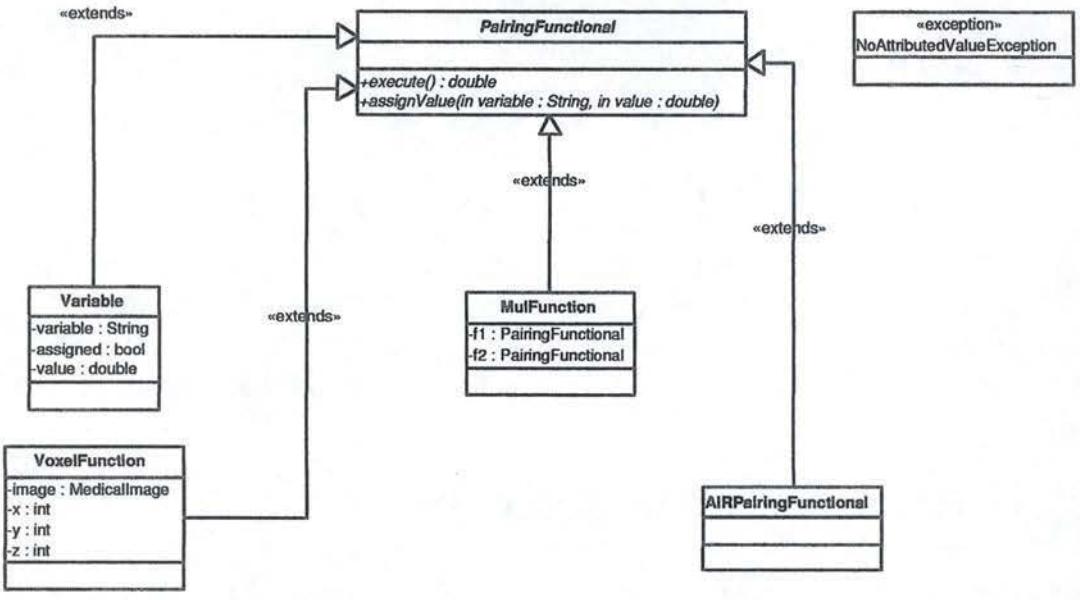


FIG. 5.6 – Représentation des fonctionnelles d'appariement

classe	représente
Value	constante
Variable	variable
VoxelFunction	un voxel
SumFunction	la somme de fonctions
MulFunction	le produit de fonctions
DivFunction	le quotient de fonctions
BigSumFunction	la grande somme Σ
BigMulFunction	le grand produit Π
Exponent	les puissances de fonction

Nous n'allons pas détailler l'ensemble des classes mais simplement émettre quelques remarques à l'attention de certaines d'entre-elles.

Tout d'abord, notons que la classe `Variable` simule la notion de classique de variable en associant à un nom de variable une valeur. C'est au sein de cette classe que l'assignation s'effectue réellement. La classe `Value` représente les constantes, la fonction `assign(variable,valeur)` n'y est donc pas implémentée. `VoxelFunction` permet quant à elle de récupérer la valeur d'un voxel à partir de sa position. Finalement, nous abordons un peu plus profondément la classe qui a pratiquement été utilisée jusqu'à présent, `AIRPairingFunctional`.

5.3.3.1 AIRPairingFunctional

Lors de l'élaboration de la plateforme, il fallait pouvoir incorporer le programme `alignlinear`. Il n'a pas été possible, durant la période de stage, d'appliquer la découpe élaborée lors de l'étude de cas portée sur *AIR*. En effet, bien qu'open source, le package est assez peu documenté et la compréhension de l'ensemble des paramètres des fonctions de réalignement aurait recquis trop de temps. C'est pourquoi le choix d'insérer `alignlinear` dans son entièreté a été posé.

L'utilisation du *JNI* implique certaines restrictions quant au format des modules C employés. Nous ne détaillerons pas ici les modifications à apporter, mais conseillerons le lecteur à parcourir le tutorial consacré au JNI, en annexe.

Nous avons intégré `alignlinear` en faisant en sorte de modifier au minimum son code source. `alignlinear` étant à l'origine une programme C exécutable, nous avons dû opérer quelques modifications quant au nom de la fonction principale. `main(int argc, char* argv)` s'est vu transformée en `main_alignlinear(int argc, char* argv)`. De plus, cet exécutable était suivi de l'ensemble des paramètres nécessaire au bon déroulement du processus (fichiers à recaler, nom du fichier de transformation en sortie, paramètres liés au threshold[2]). La ligne de commande ressemble généralement à

```
alignlinear maitre recal transform -m6 -t1 150 -t2 150
```

où

- `maitre` est le fichier modèle de la transformation (Les fichiers utilisés doivent être au format ANALYZE)
- `recal` est le fichier à recaler (idem)
- `transform` est le fichier contenant la transformation (fichier *.air*)
- `-m6` spécifie le type de transformation à rechercher, dans ce cas-ci une transformation affine
- `-t1 <nombre>` spécifie la valeur du threshold pour le fichier modèle
- `-t2 <nombre>` idem mais pour le fichier à recaler

Il fallait donc passer la même syntaxe d'arguments à `main_alignlinear` qu'à `main`. Pour ce faire, nous avons créé une structure Java, à savoir un tableau de chaînes de caractères que nous transformons en tableau de chaînes de caractères compréhensibles par un module C et ce, grâce aux méthodes mises à notre disposition par le *JNI*. Le

[2]Pour rappel, le threshold est le seuil à partir duquel les voxels sont considérés dans le processus de coregistration

lecteur intéressé trouvera en annexe le code correspondant à cette traduction de structure. Pratiquement, cette traduction est effectuée au sein de la méthode privée déclarée native `doAlignLinear`. Notons enfin que, si pour une raison de mauvaise paramétrisation du module, la recherche d'une transformation optimale n'était pas possible, une exception de type `NativeException` est levée.

Comme nous l'avons dit, nous n'avons pas incorporé la méthode `alignlinear` selon la découpe effectuée dans l'étude de cas. Si le déroulement du processus se passe bien, il en résulte un fichier *.air* contenant les informations relatives à la transformation à appliquer. Il nous faut donc seulement aller récupérer les données contenues dans ce fichier pour en faire un élément de type `Transformation`. Cette tâche est réalisée par la méthode privée native `getNativeMatrix`. Il est certain que cette modélisation n'est pas parfaite. Nous reviendrons sur ce point lors de la critique de l'architecture, dans le chapitre suivant.

5.3.4 La méthode d'optimisation

Suite à ce qui a déjà été dit au sujet de l'incorporation d'`alignlinear`, la partie concernant la méthode d'optimisation a relativement peu été développée. Néanmoins un schéma général peut être déduit de ce que nous avons déjà dit au sujet de ces méthodes dans la découpe en composantes des algorithmes de coregistration. Il convient de rappeler qu'une méthode d'optimisation spécifie la manière dont la recherche des paramètres de la transformation optimale est effectuée.

La classe abstraite `OptimisationMethod` représentera cette composante, définissant la méthode abstraite `findTransformation`, qui prendra en argument une fonctionnelle d'appariement, permettant ainsi de mesurer au cours de la recherche le degré de similitude entre les deux images, et également les caractéristiques issues des images à coregistrer. Le résultat sera la transformation optimale. La figure 5.7 reprend ces concepts.

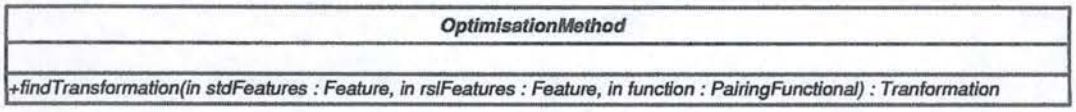


FIG. 5.7 – Représentation des méthodes d'optimisation

5.4 Couche 3 : la stratégie

La troisième couche de notre architecture définit l'ensemble des algorithmes qui seront disponibles pour pratiquer une coregistration. C'est ici qu'intervient Le Design Pattern **stratégie** que nous avons défini auparavant. Rappelons que le pattern design stratégie est de type comportemental, traitant de la collaboration d'objets pour accomplir une tâche. En l'occurence, le modèle stratégie définit une famille d'algorithmes par le biais d'une interface commune, qui peuvent ainsi être interchangeables dynamiquement et évoluer indépendamment des clients qui les utilisent.

L'implémentation que nous en avons faite se base dès lors sur ce modèle. Nous retrouverons une classe équivalente à la composante *Context* du modèle, `RegistrationContext`. Cette classe servira à maintenir une référence vers la stratégie de coregistration adoptée. On y trouvera également les références vers les images médicales concernées, ainsi que vers l'image résultant du processus de coregistration. Enfin, outre la fonctionnelle d'appariement et la méthode d'optimisation, on retrouvera les paramètres de pilotage des algorithmes. La présence de ces paramètres a été dictée par les besoins de paramétrisation du package *AIR*. Néanmoins, il ne semble pas faux de penser que la plupart des algorithmes de coregistration nécessiteront une certaine paramétrisation.

La classe abstraite `RegistrationStrategy` sera l'équivalent de la classe *Strategy* du modèle stratégie. Elle posera la signature des méthodes que toute implémentation concrète devra respecter. Nous y avons déclaré

- `doRegistration(MedicalImage im1, MedicalImage im2, PairingFunctional f, OptimisationMethod o) : void`
  Cette méthode va mener le processus de coregistration à son terme. Elle appelera les trois autres méthodes de la classe. La sémantique des autres fonctions nous permettra de dresser son pseudo-code

- `researchTransformation (Feature feat1, Feature feat2, PairingFunctionnal f, OptimisationMethod o) : Transformation`
  Celle-ci va s'occuper à proprement dit de la recherche de la transformation. Le coeur de l'algorithme s'y trouve. C'est elle qui utilisera les fonctionnelles d'appariement et les méthode d'optimisation

- `extractFeature(MedicalImage img) : Feature`
  Cette méthode va extraire les caractéristiques de l'image médicale

- `ApplyTransformationOnSecondImage(MedicalImage img,`

`Transformation trans) : MedicalImage`
Cette dernière va appliquer la transformation trouvée sur l'image à recaler

Le schéma 5.8 illustre la modélisation adoptée.

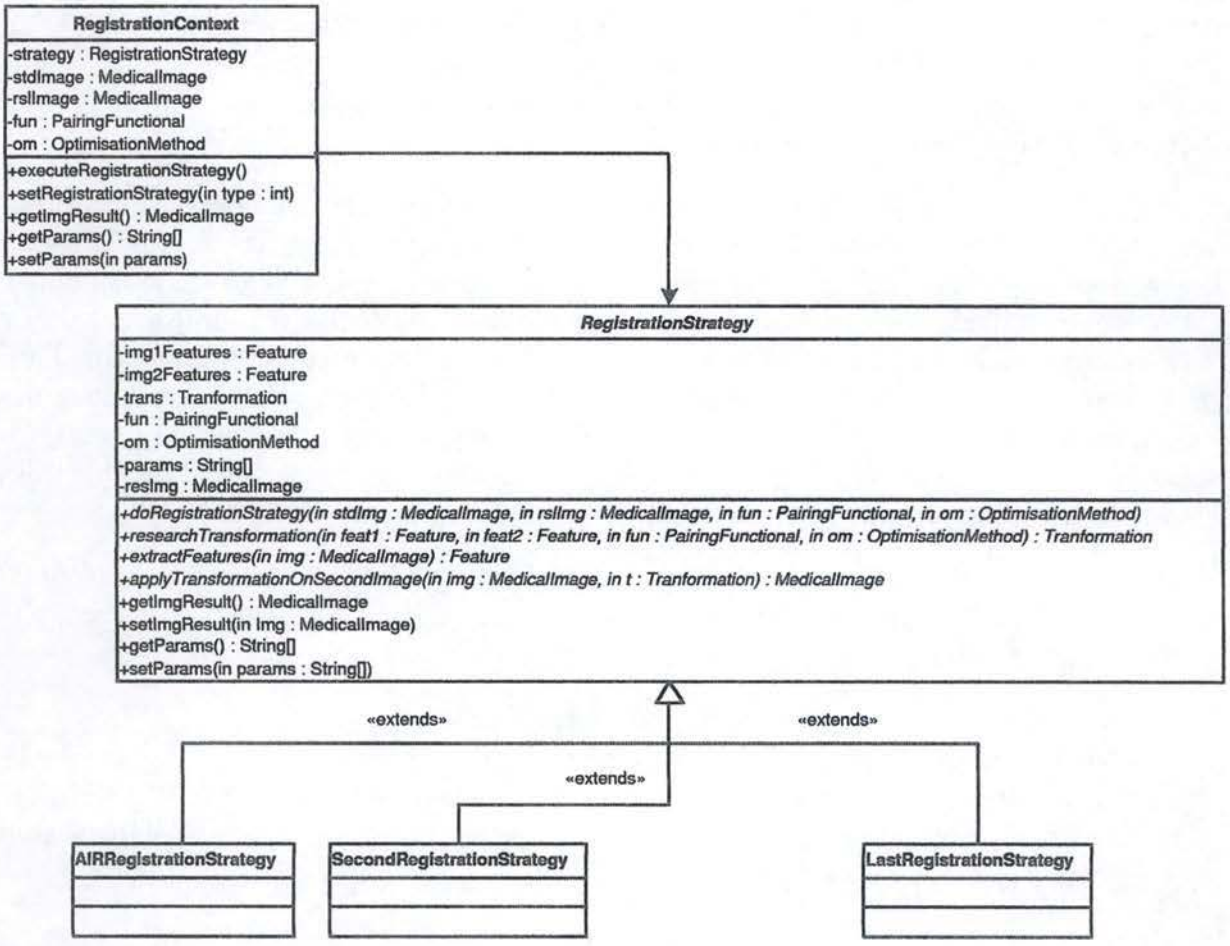


FIG. 5.8 – Modélisation de la couche stratégie

Au vue de ce qui a été dit sur la sémantique des méthodes, on peut aisément extrapoler un pseudo-code pour la méthode `doRegistration()`.

```
doRegistration(MedicalImage img1, MedicalImage img2,
PairingFunctional f, OptimisationMethod o){

        Feature feat1 = extractFeatures(img1);
        Feature feat2 = extractFeatures(img2);
        Transformation t = researchTransformation(feat1, feat2, f, o);
        MedicalImage imgRes = applyTransformationOnSecondImage(img2,t);
}
```

où `imgRes` est un attribut de la classe abstraite `RegistrationStrategy`.

A la vue de ce code, il aurait été possible de faire de la méthode `doRegistration()` une méthode concrète plutôt qu'abstraite. Ce choix a été dicté par des raisons de liberté d'implémentation. En effet, comme nous le verrons dans le paragraphe suivant, des actions qui sortent de la sémantique de chacune des méthodes peuvent être entreprises. C'est pourquoi nous avons préféré laisser `doRegistration()` abstraite.

5.4.1 AIRRegistrationStrategy

Seule stratégie de coregistration concrète, elle ne respecte malheureusement pas le schéma mis en évidence ci-dessus. En effet, de par le fait qu'elle ait été incluse dans son entièreté, certains composants n'étaient d'aucune utilité. Tandis qu'`AIRPairingFunctional` reprend l'ensemble du processus de recherche de la transformation, la méthode d'optimisation est inutile. Il est certain qu'avec du recul, il aurait été plus logique d'intégrer l'ensemble de l'algorithme à ce niveau plutôt qu'à la couche 2. Quoiqu'il en soit, une amélioration de la plateforme sera en effet d'effectuer une découpe d'*AIR* de telle manière qu'elle respecte la sémantique proposée. Nous allons à présent passer en revue les méthodes principales de cette classe.

5.4.1.1 extractFeatures

Nous avons vu dans la partie décrivant le package *AIR* que celui-ci acceptait un format d'image bien spécifique. Il a donc fallu mettre au point des convertisseurs capables de traduire notre schéma de représentation interne en format *ANALYZE*. Ceci est l'objet de cette fonction. Cette traduction de format est réalisée à l'aide d'outils développés à cette occasion. Nous reviendrons sur ceux-ci dans la section consacrée aux outils.

5.4.1.2 researchTransformation

Bien que cette fonction prenne en paramètre les caractéristiques issues des images, celles-ci ne sont pas utilisées car, comme nous l'avons dit plus haut, `alignlinear` exige des fichiers dans un format spécifique. A terme, il serait intéressant de modifier l'algorithme en C de telle manière qu'on ne doive plus passer par des fichiers *ANALYZE*, car ces opérations de codage et décodage sont gourmandes en ressources. La méthode d'optimisation n'est donc, elle non plus, pas utilisée. Seule la fonctionnelle d'appariement est utilisée, par le biais de la méthode `execute()` appliquée sur l'instance de `PairingFunctional`, qui fait appel à la fonction en code natif `alignlinear`.

Cette dernière fonction prend en argument certains paramètres, comme nous l'avons vu dans la section décrivant l'implémentation de la classe `AIRPairingFunctional`. Ces arguments seront fournis par la couche supérieure et devront être interprétés correctement par `alignlinear`. Des méthodes privées ont été implémentées pour permettre à ce dernier de comprendre correctement ces paramètres.

5.4.1.3 applyTransformationOnSecondImage

Dans cette méthode, nous ferons une seconde fois appel au *JNI*. En effet, comme indiqué sur le schéma 3.4, une fois la transformation optimale trouvée, il faut encore l'appliquer sur l'image à recaler. Le package *AIR* met pour cela à notre disposition la routine `reslice`. Outre l'appel à la fonction C, cette méthode aura aussi pour mission de charger dans la représentation interne le résultat produit par la-dite fonction.

5.4.1.4 doRegistration

Comme explicité dans le pseudo-code, cette méthode fera appel aux méthodes décrites ci-dessus. En plus de celles-ci, elle s'attachera à nettoyer du disque les différents fichiers temporaires créés lors de la conversion en *ANALYZE*.

5.4.2 Combinaison

Le véritable attrait de la plateforme se situe au niveau de cette couche. Les diverses implémentations de ces méthodes dans chacune des stratégies concrètes sont susceptibles d'offrir un large panel d'algorithmes. En outre, ce panel se verra décuplé par l'interchangeabilité des différentes méthodes d'optimisation et fonctionnelles d'appariement.

L'ensemble de ces combinaisons permettra à l'utilisateur de bénéficier d'un large éventail de méthodes pour mener à bien sa coregistration.

5.5 Couche 4 : l'éditeur

Ultime couche de la plateforme, elle est attachée à la gestion des fonctions de bases du programme (ouverture, fermeture et sauvegarde de fichiers), à la gestion de l'environnement visuel ainsi qu'à la sélection des stratégies. Cette couche est donc scindée en trois composantes, à savoir le système de gestion de l'affichage des images, le système de gestion des stratégies et l'élément charnière, l'éditeur en lui-même.

Nous commencerons par décrire ce dernier. `CoregInterface` est le point d'entrée du programme. Il se présente graphiquement comme une simple fenêtre[3] à partir de laquelle sont exécutées les différentes actions pour mener à bien les processus de coregistration.

On y retrouve les différentes méthodes d'ouverture et de sauvegarde de fichiers. Remarquons qu'il est possible de sauvegarder les fichiers modèle et à recaler, pour la simple raison que des modifications peuvent être effectuées dessus, notamment des opérations de préprocessing[4]. La sauvegarde d'un fichier sous un nom déjà existant a, quant à lui, été interdit. En effet, les données médicales sont extrêmement précieuses, et il aurait été ennuyeux qu'une erreur de manipulation entraîne la disparition d'une de ces données originales. L'ouverture et la sauvegarde des fichiers a nécessité l'implémentation d'encodeurs et décodeurs DICOM, facilitée par l'utilisation des packages *emim.jar* et *zdicom.jar*.

Cet éditeur doit également pouvoir gérer la sélection des stratégies parmi l'ensemble des stratégies possibles. Cette tâche est attribuée au gestionnaire des stratégies, modélisée par la classe `ChooseCoregistrationStrategy`. Lorsqu'une stratégie concrète a été choisie, il devrait être possible de choisir via une interface graphique les différentes fonctionnelles et méthodes d'optimisation existantes. Actuellement, étant donné qu'aucune fonctionnelle et méthode d'optimisation ne peut être utilisée comme telle, il n'a pas été prévu d'interface pour réaliser ce choix.

Une fois l'algorithme de coregistration sélectionné, et en supposant que l'utilisateur ait bien chargé une image modèle et une image à recaler, il est possible de démarrer le processus de coregistration. En fonction de l'algorithme sélectionné (le choix d'un algorithme déclenche la méthode `setRegistrationStrategy()` au niveau de la classe

[3]Des captures d'écran sont disponibles en annexe, dans la section relative au mode d'emploi de la plateforme

[4]Nous reviendrons sur celles-ci dans la section consacrée aux outils

`CoregInterface`), une interface de configuration apparaît, permettant de configurer les différents paramètres de la fonction utilisée (dans notre cas `AIRConfiguration`).

Une fois l'algorithme paramétré, la méthode `executeRegistration()` de l'éditeur est appelée. Celle-ci crée le contexte de coregistration en lui adjuvant les différentes images médicales, la fonctionnelle d'appariement, la méthode d'optimisation et la stratégie concrète sélectionnée, à laquelle sont associés les paramètres du module de configuration. Il est à noter qu'il aurait été plus adéquat de créer un contexte de coregistration lors de l'initialisation du programme, en modifiant seulement ses attributs (stratégies, fonctionnelles, ...) en fonction des besoins, au lieu d'en recréer un à chaque lancement de processus. Le processus de coregistration se lance finalement, via la méthode `doRegistration()` de la classe héritant de `RegistrationStrategy` (abordé à la section précédente). Si tout se passe correctement (un mauvais paramétrage pouvant rendre impossible la recherche de la transformation), l'image résultat apparaît.

Parlons justement de l'affichage des images. Plusieurs classes ont été implémentées à cet effet. La classe `ImageRendering` est le gestionnaire central. Elle s'appuie sur deux autres classes. La première, `ImageInfoInterface`, permet d'afficher des informations relatives aux caractéristiques de base de l'image, modélisée par la classe `BasicImageInfo` de la couche 1. La seconde, `SliceRendering` s'occupe de l'affichage[5] d'une image plane (`MedicalImage2D`). `ImageRendering` s'occupe de la coordination de l'affichage des différentes tranches composant une image volumique[6]. La méthode `updateImage()` peut être utilisée dans le cadre de manipulations ne modifiant pas les caractéristiques de l'image (par exemple effectuer une translation du volume dans l'espace[7]). Notons que c'est la classe `CoregInterface` qui gère la mise à jour des images via les fonctions `updateDisplay()` et `newDisplay()`.

Finalement, la figure 5.9 illustre ce fonctionnement.

Ajoutons pour terminer que les méthodes gérant l'interdépendance entre les différentes interfaces ont été volontairement omises, inutiles dans le cadre de la description de l'architecture.

[5]Pratiquement, l'affichage se fait au sein d'un *JPanel* pour lequel la méthode *paintComponent* a été surchargée

[6]Rappelons qu'une image volumique, représentée par `MedicalImageStack`, est composée d'un ensemble d'images planes `MedicalImage2D`

[7]L'outil *Mover* a été développé à cet effet, nous en reparlons dans la section consacrée aux outils

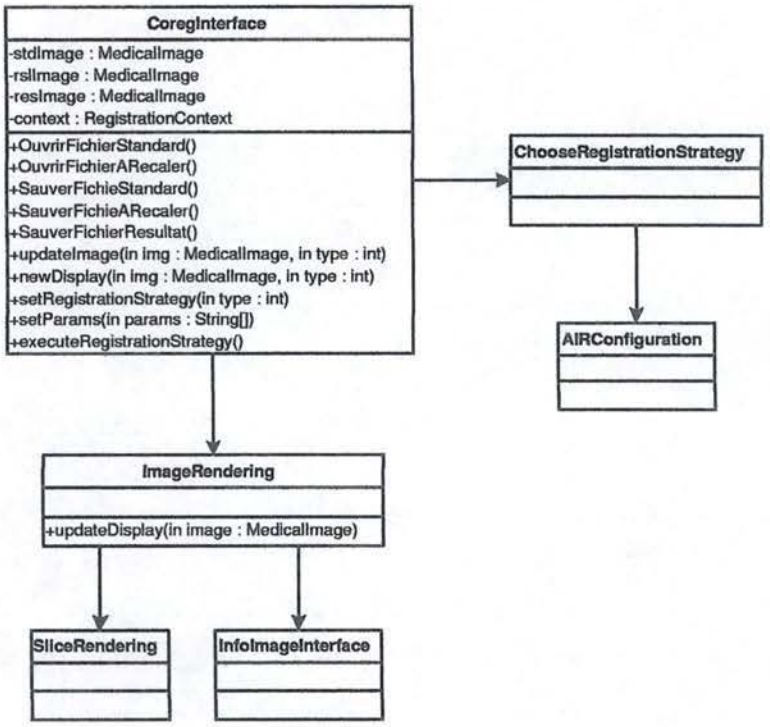


FIG. 5.9 – Représentation de la couche 4

5.6 Vue d'ensemble de l'architecture

Le schéma 5.10 illustre de manière simplifiée l'ensemble des différentes couches de l'architecture. Toutes les relations entre les différentes entités n'ont pas été représentées, évitant ainsi une surcharge du schéma.

5.7 Les outils annexes

A côté de l'architecture expliquée dans les sections précédentes, des outils complémentaires ont été implémentés. D'une part, un ensemble d'encodeurs et décodeurs ont été développés, nécessaires non seulement pour communiquer avec le package AIR, mais également pour pouvoir comprendre le format DICOM. D'autre part, des outils pour la coregistration et de traitement des images ont également été réalisés. Cette section abordera ces diverses productions.

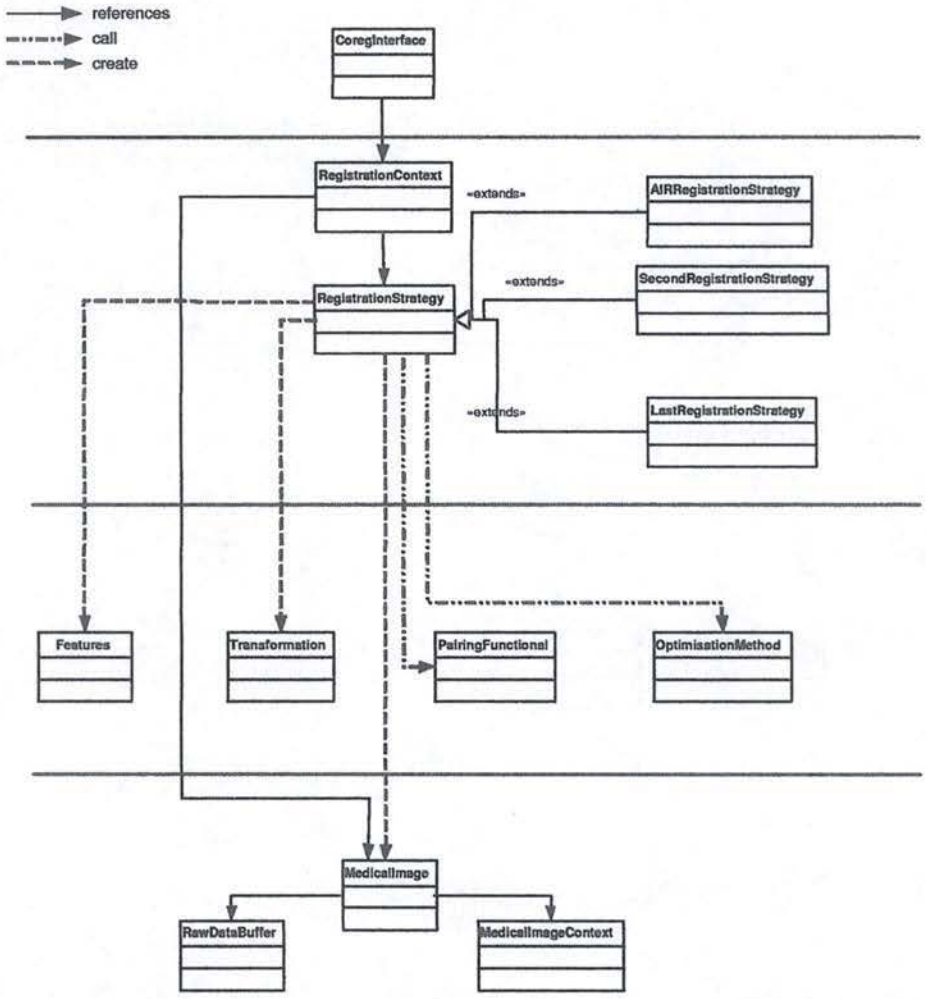


FIG. 5.10 – Vue d'ensemble de l'architecture

5.7.1 Encodeurs et décodeurs

5.7.1.1 Encodeurs

Deux classes ont été prévues pour répondre à ces besoins. Tout d'abord la classe `ToAIR` qui permet simplement de convertir une image de type `MedicalImage` en un fichier compatible *ANALYZE*, c'est-à-dire en créant un fichier d'en-tête *.hdr* contenant toutes les informations techniques relatives à l'image, et un fichier de données brutes, tel qu'expliqué dans le chapitre 3.

Le deuxième encodeur a été créé pour permettre la sauvegarde d'une image au format *DICOM* à partir d'une image `MedicalImage`. Il faut se rappeler que chaque image *DICOM* contient, outre des données brutes, un ensemble de données relatives non seulement à l'image, mais également au patient, à l'examen etc... Lorsqu'une image est recalée par rapport à une autre, la majorité de ces informations reste inchangée. Seules les données techniques relatives à l'image sont modifiées. Par le biais du package *zdicom* développé aux Cliniques Universitaires de Mont-Godinne, il a été possible de récupérer ces données et de les modifier afin de conserver la nouvelle image sur un support non volatile. La classe `DICOMDataObjectOperations` a été créée à cet effet.

5.7.1.2 Décodeurs

Comme pour les encodeurs, les décodeurs ont été réalisés pour supporter les deux formats utilisés. Le premier, `AIROpener` permet d'ouvrir les fichiers au format *ANALYZE*, en analysant tout d'abord le fichier d'en-tête, et ensuite, en fonction des informations qu'il aura trouvé dans celui-ci, en lisant le fichier de données brutes.

Le décodeur utilisé pour les fichiers *DICOM* se devait d'être légèrement différent du précédent dans le sens où une image volumique peut revêtir différentes formes. Elle peut se présenter soit sous la forme d'un fichier unique, contenant l'entièreté des informations, soit sous la forme d'une multitude de fichiers, où chacun d'entre eux représente une seule tranche du volume (y compris les données administratives). Deux classes ont alors été développées, l'une permettant d'ouvrir les fichiers simples, l'autre les fichiers multiples (respectivement `DICOMOpener` et `DICOMsOpener`).

5.7.2 Outils dédiés à la coregistration

A côté de l'intégration d'*AIR*, deux outils ont été implémentés pour aider l'utilisateur dans sa tâche.

Le premier, appelé *Mover*, permet d'effectuer des translations d'images. Il peut déplacer le volume dans deux directions, à savoir suivant l'axe des X ou selon l'axe des Y. Il est ainsi possible d'effectuer ainsi un premier recalage manuel, assez facile d'accès pour l'utilisateur (il peut régler simplement les pas de déplacement de l'image et annuler toute modification si besoin est). Dans le cas de deux images fort décentrées l'une par rapport à l'autre, ce premier recalage manuel permettra certainement à l'algorithme de trouver la solution optimale plus rapidement.

Le second, *Zoomer*, permet de rendre un volume isotrope, c'est-à-dire dont les voxels

sont cubiques. Pratiquement, cette transformation est réalisée pau moyen d'une interpolation de l'image de telle sorte que les voxels aient comme dimension la plus petite des dimensions du voxel original. Il a été implémenté en utilisant le *JNI*. Il exploite un outil disponible dans le package *AIR*.

Conclusion

Nous arrivons peu à peu au bout de ce mémoire. S'il fallait retenir trois grandes idées après le parcours de ce chapitre, nous citerions

1. **La découpe en couches** qui permet d'appréhender au mieux la sémantique des divers composants et leurs relations. Nous avons ainsi modélisé
   - une première couche dédiée à la représentation des images médicales
   - une seconde couche représentant les quatre caractéristiques communes aux algorithmes de coregistration
   - une troisième couche dite de stratégie, permettant l'encapsulation de différents algorithmes de coregistration
   - une dernière couche gérant l'environnement graphique et les opérations de base.
2. **la présence d'une couche stratégie** qui permettra à l'utilisateur de spécifier le type d'algorithme qu'il désire employer pour effectuer son recalage.
3. **la modélisation des caractéristiques communes des algorithmes** de coregistration, dont la modélisation générique devrait permettre une interchangeabilité des fonctionnelles d'appariement et des méthodes d'optimisation au niveau de la couche stratégie.

Nous terminerons le présent mémoire par un chapitre consacré à la critique de cette réalisation et nous émettrons des avis quant aux futures améliorations et réalisations de cette plateforme.

Chapitre 6

Critiques

Après avoir abordé longuement la description de l'architecture et son implémentation, nous allons à présent discuter de l'état actuel du développement de la plateforme et des modifications à y apporter.

6.1 Résultats

Nous allons présenter les résultats liés à l'implémentation de la plateforme selon trois angles, aussi essentiels les uns que les autres. Nous allons tout d'abord analyser les résultats obtenus par rapport à **l'intégration du programme au sein de son environnement**, ensuite nous aborderons **l'intégration du package *AIR*** et enfin nous évaluerons **la présentation des résultats**, essentielle pour ce type de programme.

6.1.1 Intégration dans son environnement

Il fallait que le programme s'insère parfaitement par rapport au format d'image utilisé aux Cliniques Universitaires UCL de Mont-Godinne, à savoir *DICOM*. Des encodeurs et décodeurs ont donc été développés. Jusqu'à présent, ils ont montré entière satisfaction. En effet, les images ont pu être chargées à l'intérieur du programme, modifiées et puis ensuite sauvegardées. Les images ainsi sauvées ont pu être lues avec des viewer spécialisés dans la lecture d'image DICOM. On peut donc affirmer que cette intégration s'est bien passée.

6.1.2 Intégration du package *AIR*

Par l'utilisation du *Java Native Interface*, la plateforme a pu s'enrichir de trois programmes dédiés à la coregistration. Nous avons longuement abordé *alinglinear*, l'outil principal pour recaler deux images. Nous avons également parlé de *reslice*, permettant d'appliquer la transformation sur l'image. Enfin, nous avons incorporé *zoomer*, pour rendre les voxels d'une image cubiques.

L'insertion des deux derniers et leur utilisation est satisfaisante. Autant *reslice* que *zoomer* fonctionnent parfaitement au sein de la plateforme. Malheureusement, les résultats obtenus avec *alignlinear* sont moins convaincants. En effet, l'algorithme a du mal à coregistrer les images. S'il parvient à harmoniser la taille des voxels par rapport à l'image modèle, il n'arrive pas à calculer correctement les paramètres de translation et de rotation. Le problème doit certainement venir de l'encodeur utilisé pour convertir la représentation des images en format *ANALYZE*. En effet, il semble que le module de coregistration ne parvient pas à interpréter correctement les valeurs des voxels. Peut-être les données contenues dans le fichier en-tête *.hdr* ne sont pas adéquates. Quoiqu'il en soit, les résultats obtenus à ce propos ne sont pas encore suffisants.

6.1.3 Présentation des résultats

Comme nous l'avons souligné dans le premier chapitre, la présentation des résultats de la coregistration est primordiale pour permettre aux médecins de fournir un diagnostique. Le système de présentation aujourd'hui implémenté[1] consiste en l'affichage simple des images dans une fenêtre. Si un marqueur visuel (activable ou non) peut aider le médecin dans son analyse des clichés, les méthodes d'affichage se révèlent assez pauvres. Elles consistent simplement à afficher les images, sans technique particulière (parallel display ou autre). Ce n'est certainement pas assez pour permettre l'établissement aisé d'un diagnostique. D'énormes travaux devront être fait à ce sujet. Nous en reparlerons dans la section consacrée aux améliorations à apporter.

6.2 Modifications

Nous allons maintenant aborder les modifications à apporter à l'implémentation actuelle de la plateforme. En terme de structure, certaines redondances ont été introduites

[1] Des images de l'interface sont disponibles en annexe, dans la section consacrée à l'utilisation de la plateforme.

au niveau de la classe abstraite `RegistrationStrategy` par rapport au contenu de la classe `RegistrationContext`, notamment au niveau des fonctionnelles d'appariement et méthodes d'optimisation. En effet, de part le paramétrage des méthodes déclarées dans `RegistrationStrategy`, ces attributs peuvent être simplement supprimés.

Une seconde modification, dont nous avons déjà beaucoup parlé au travers de ce mémoire, est la découpe d'*alignlinear*. Cette découpe sera certainement facilitée par le fait que les fonctions mises en évidence dans le chapitre quatre font l'objet de fichiers séparés, dans le package *AIR*. Néanmoins, il est certain que ce ne sera pas un travail évident, étant donné le nombre de paramètres à ses fonctions. On pourrait également envisager d'essayer de court-circuiter la création de fichiers temporaires au format *ANALYZE* en fournissant directement le format d'image nécessaire à l'algorithme, et récupérer le résultat de la coregistration, toujours sans passer par des fichiers temporaires.

Enfin, une dernière modification proposée serait de modifier quelque peu la structure au niveau de la gestion des fenêtres, afin d'avoir toujours une seule instance d'un panneau de configuration d'un algorithme ou autre. Actuellement, la gestion des instances uniques n'est pas réalisée. Cette gestion pourrait s'exprimer par exemple via le Design Pattern **Singleton**, qui permet justement cette unicité des instances de classes.

Conclusion

Nous pouvons voir à travers ce chapitre que la plateforme de coregistration n'est pas encore opérationnelle en tant que telle. Bien qu'elle soit relativement bien intégrée dans son environnement, des modifications sont nécessaires pour qu'elle puisse fournir des résultats intéressants.

Conclusion

Dans ce mémoire, nous avons dans un premier temps abordé des notions essentielles liées au monde de la coregistration d'images médicales. Après avoir brièvement introduit le sujet, nous nous sommes attachés à définir le concept d'image médicale. Après cela, nous avons pu distinguer les deux grands types d'images médicales à travers la notion de modalité. A travers celle-ci nous tentions de faire sentir les difficultés d'une coregistration mentale. Cette coregistration faisait alors l'objet de la section suivante. Après avoir classifié les familles de coregistration, nous avons mis l'accent sur le côté présentation des résultats de cette coregistration. En effet, cet aspect est primordial dans le cadre d'une plateforme dédiée à la coregistration d'images médicales. Le médecin doit pouvoir disposer d'outils de visualisation adéquats pour évaluer la qualité du recalage. Finalement, nous tentions de faire sentir intuitivement les différentes composantes de tout algorithme de coregistration.

Le deuxième chapitre abordait plus en détail chacune des ces composantes. Nous y avons exposé l'espace des caractéristiques, représentant l'ensemble des points de repère sur lesquels se base le processus de coregistration. La deuxième composante abordée était la transformation, définissant un ensemble de paramètres de transformation. Nous explicitions ensuite la fonctionnelle d'appariement, permettant de mesurer le degré de similitude entre deux images. Enfin, la dernière composante était la méthode d'optimisation, stipulant la manière dont la recherche de la transformation optimale est effectuée. Nous avons abordé ces composantes sans rentrer dans les détails mais suffisamment que pour comprendre la modélisation qui en sera faite au sein de la plateforme.

Le troisième chapitre quant à lui discutait des différentes technologies et outils utilisés dans la réalisation de la plateforme. Il présentait notamment le package *AIR*, ensemble d'outils écrits en C dédiés à la coregistration d'images médicales. Un de ces outils allait être utilisé au sein de la plateforme, *alignlinear*.

Cet outil, permettant de calculer les paramètres de transformation pour qu'une image corresponde au mieux à une autre, a fait l'objet d'une étude dans le chapitre suivant. Nous avons identifié les différents composants exprimés au chapitre deux et ce, à dessein

de l'insérer au mieux dans la plateforme.

Le chapitre cinq est le point central du mémoire car nous y avons présenté l'architecture de cette plateforme. La découpe en couche y est abordée et l'on y retrouve les différentes notions que nous avons explicitées aux chapitres précédents. La première couche modélise la notion d'image médicale, la seconde les quatre composantes de tout algorithme de coregistration, la troisième gère justement ces algorithmes et la dernière gère entre autre la présentation des résultats.

Finalement, nous avons, au travers du dernier chapitre, critiqué l'état d'avancement de la plateforme. Nous y avons vu que tout n'était pas encore opérationnel et qu'énormement de travail était encore nécessaire pour qu'elle puisse être totalement utilisable dans son rôle d'aide au diagnostique.

Néanmoins, le développement de la plateforme ne doit pas s'arrêter à la réalisation de ces quelques modifications. Si elles amélioreront sensiblement son fonctionnement, elles ne lui permettront cependant pas d'être utilisée à des fins de diagnostique. Un réel effort doit être fait au niveau de la présentation des résultats. Il serait intéressant de mettre en place une architecture de gestion de la présentation permettant de basculer d'un mode de présentation à un autre. Ces modes de visualisation ont été quelque peu abordés au cours du premier chapitre et il semble clair qu'il faille fournir un tel outil à la plateforme.

Enfin, il serait intéressant de doter la plateforme d'un système permettant à l'utilisateur de l'aider dans sa paramétrisation des algorithmes. Celle-ci peut s'avérer particulièrement fastidieuse. Lorsque nous avons abordé la paramétrisation d'`alignlinear`, nous nous sommes restreints aux paramètres essentiels. Pratiquement, plus de vingt paramètres différents sont disponibles. Pour guider l'utilisateur, plusieurs systèmes peuvent être envisagés, des plus simples aux plus complexes. On pourrait dans un premier temps conserver les réglages utilisés dans des fichiers de configuration de type *XML*, en sauvant ainsi des typologies de configuration suivant le type d'images. On pourrait également aller plus loin en proposant une sorte de système expert, capable, en fonction du type d'image et des caractéristiques de celle-ci, de proposer une configuration type, orientant ainsi l'utilisateur dans sa paramétrisation.

Le développement de la plateforme n'en est donc qu'à ses débuts. De nombreux travaux et recherches devront encore être fournis pour qu'elle puisse déployer toute sa valeur.

Bibliographie

[CPA99] P. Cachier, X. Pennec, and N. Ayache. *Fast non rigid matching by gradient descent : study and improvements of the Demons algorithms.* Technical report, INRIA - Institut National de Recherche en Informatique et en Automatique, 1999.
http ://www.inria.fr/rrrt/rr-3706.html.

[Des00] A. Descartes. *Java Native Methods*, 2000.
http ://www.symbolstone.org/technology/java/nmbook/JNM_full.pdf.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[HC00] Cay S. Hostmann and Gary Cornell. *Au coeur de Java 2 - fonctions avancées.* Sun Microsystems press, 2000.

[KAPF97] J. Kiebel, J. Ashburner, J-B. Poline, and K. J. Friston. *MRI and Pet Coregistration - A crossvalidation of SPM and AIR.* Technical report, Departement of Neurology, Friedrich-Schiller-University, Jena, Germany - The Welcome Departement of Cognitive Neurology, The Institue of Neurology, London, United Kingdom, 1997.
http ://www.fil.ion.ucl.ac.uk/spm/papers/SJK_coreg/SJK_coreg.ps.gz.

[KFM+99] T. Vu Khac, J. Fichefet, H. Meurisse, J-P. Leclerq, F. Vandemeersch, A. de Rosée, O.Manirakiza, and S. Dury. *Review and comparison of coregistration methods.* Technical report, Institut d'Informatique de Namur et Cliniques Universitaires U.C.L. de Mont-Godinne, 1999.

[MV] J. B. A. Maintz and M. A. Viergever. *An Overview of Medical Image Registration Methods.*
http ://www.cs.uu.nl/people/twan/personal/brussel_bvz.pdf.

[Pen96] Xavier Pennec. *L'incertitude dans les problèmes de Reconnaissance et de Recalage – Applications en Imagerie Médicale et Biologie Moléculaire.* PhD thesis, Ecole Polytechnique, Palaiseau, France, 1996.
http ://www-sop.inria.fr/epidaure/BIBLIO/Author/PENNEC-X.html.

[Roc01] Alexis Roche. *Recalage d'images médicales par inférence statistique.* PhD thesis, Université de Nice Sophia-Antipolis, 2001.
http ://www-sop.inria.fr/epidaure/BIBLIO/Author/ROCHE-A.html.

[Sey02] Stéphane Seynave. *Coregistration d'images médicales : contexte, modélisation du problème, et sa traduction en une première conception Orientée Objets.* Master's thesis, Facultés Universtaires Notre-Dame de la Paix, Namur, 2002.

[Van98] François Vandermeersch. *Présentation multimodale en imagerie médicale.* Master's thesis, Facultés Universtaires Notre-Dame de la Paix, Namur, 1998.

[Weba] Air - automated image registration
http ://www.loni.ucla.edu/ncrr/software/air.html.

[Webb] Air files types
http ://dns1.bmap.ucla.edu :16080/air5/file_types.html.

[Webc] http ://www.brainvoyager.com/bv2000onlinehelp/brainvoyagerwebhelp/mergedprojects/coregistration/manual_interactive_2d_3d_coregistration.html.

[Webd] Imagerie biologique et médicale
http ://perso.club-internet.fr/visionaute/visionartificielle/va_medical.html.

[Webe] Irm
http ://www.clinique-radiologique.com/tpe2.htm.

[Webf] Les grandes etapes de l'imagerie medicale
http ://www.montchoisi.ch/radiologie/histoirerad.html.

[Webg] Les transformations géométriques du plan
http ://www.inrialpes.fr/movi/people/boyer/teaching/maitrise/c3.pdf.

[Webh] Matrices et vecteurs - rappels sur les matrices et vecteurs appliqués à la 3d
http ://membres.lycos.fr/nicolasgate/rappels.htm.

[Webi] Multi modality volume registration (mmvreg v1.0b) user manual
http ://noodle.med.yale.edu/ cs/software/mmvreg/mmvreg.html.

[Webj] Overview of the jni
http ://java.sun.com/docs/books/tutorial/native1.1/concepts/index.html.

[Webk] Qu'est ce que l'image numérique ?
http ://www.cddp74.edres74.ac-grenoble.fr/article.php3 ?id_article=5.

[Webl] Rappels sur les transformations et les coordonnées homogènes
http ://www.gpa.etsmtl.ca/cours/gpa669/lectures-powerpoint/lecture 9.ppt.

Annexe 1 : Java Native Interface

Intérêt

Le JNI fournit une interface de programmation générique (par rapport à l'implémentation de chaque machine virtuelle Java) pour le codage des méthodes natives. Cette généricité assure une totale portabilité à travers les différentes plates-formes. Il permet donc à un code Java d'interagir avec des librairies écrites dans un autre langage tel que C ou C++. Cette interaction va dans les deux sens, autrement dit, JNI permet à des programmes Java de faire appel à des méthodes natives (ce qui nous intéresse plus particulièrement) ou bien d'utiliser des méthodes et objets Java au sein de méthodes C ou C++.

Méthode de déploiement pour un programme Java faisant appel à des méthodes natives

Il se déroule en 5 phases :

1. Ecriture d'une classe Java déclarant les méthodes natives, ainsi que le `main`
2. Compilation de cette classe Java
3. Génération d'un fichier d'en-tête pour la méthode native, on obtient donc la signature formelle de la méthode à implémenter
4. Implémentation de cette méthode
5. Compilation dans une librairie dynamique
6. Exécution du code Java

Après avoir décrit ces différentes phases, nous aborderons un exemple concret.

Création d'une classe Java et déclaration des méthodes natives

Au sein de la classe concernée, on définira, en plus des éventuels méthodes et attributs, les signatures des différentes méthodes natives que l'on emploiera. La syntaxe est de la forme

```
[public] native [static] type maMéthodeNative(mesParamètres);}
```

Le code de cette méthode sera compilée dans une librairie dynamique (.dll sous windows, .so sous Unix). Il faudra donc spécifier dans la classe java cette librairie (sans son extension).

```
static {
      System.loadLibrary("ma_librairie");
}
```

Nous désirerons ensuite appeler cette méthode native. Pour cela, il suffira simplement de créer une instance de la classe (dans le cas d'une méthode non statique) et d'appeler la méthode sur l'objet créé.

```
monInstanceDeClasse.maMéthodeNative(mesParamètresEffectifs);
```

Compilation du code Java

Il suffit d'utiliser la commande suivante[2] pour compiler votre ficher et donc obtenir un fichier .class

```
Java monFichier.java
```

Création du fichier .h

L'utilisation de *javah* permet, à partir du fichier compilé, d'obtenir la signature formelle de la méthode native. Ce fichier d'en-tête s'obtient par la commande suivante :

```
Javah maClasse
```

Parmi d'autres déclarations, on peut y voir la signature de la méthode native que l'on devra implémenter. Elle est de la forme

```
JNIEXPORT type JNICALL Java_maClasse_maMéthodeNative(
                JNIenv*,
                JObject,
                MesParams) ;
```

[2]En supposant que la variable Classpath ait été correctement initialisée

On peut observer que la signature contient deux paramètres en plus que la signature originale de la méthode native déclarée dans le fichier java initial.

Implémentation de la méthode

Décrivons à présent les différents paramètres de la signature de la méthode à implémenter.

JNIenv * Il s'agit du pointeur d'interface JNI. Il pointe sur une table de pointeurs vers les différentes fonctions du JNI, implémentées dans chaque machine virtuelle.

JObject Cet argument diffère qu'il s'agisse d'une méthode statique ou d'une méthode d'instance. Dans ce dernier cas, il s'agit d'une référence vers l'objet sur lequel cette méthode est invoquée. Dans le cas d'une méthode statique, il s'agit d'une référence vers la classe où la méthode est définie.

MesParams Le JNI définit un ensemble de types C et C++ qui correspondent aux types dans les langages de programmation Java. Il existe deux genres de type en Java. Les types dit primitifs (tels que int, boolean, char, ...), et les types références comme les classes, les instances ou les tableaux (notons qu'en Java, le type String est une instance de la classe java.lang.String).

Le mapping des types primitifs est direct. Ainsi, le type primitif int en java devient jint en C / C++, alors que le type char de vient jchar.

Le tableau suivant regroupe les différents types primitifs en Java, ainsi que leur correspondant en C / C++

Types Java	Types JNI
boolean	jboolean
byte	jbyte
char	jchar
short	jshort
int	jint
long	jlong
float	jfloat
double	jdouble
void	void

Le mapping pour les types références n'est quant à lui pas direct. Le JNI passe des objets aux méthodes natives via des références opaques. Ce sont des types de pointeurs C qui réfèrent aux structures de données internes à la machine virtuelle Java. Pour pouvoir manipuler ces objets, il convient de faire appel aux fonctions adéquates définies dans le JNI. Par exemple, le type JNI correspondant à un String en Java, est un jstring. Pour pouvoir accéder à la valeur d'un jstring, il faut passer par les méthodes définies par le JNI, dans ce cas-ci la méthode GetStringUTFChars. (pour une spécification complète des différentes méthodes proposées par le JNI, voir site de sun : `http://java.sun.com/products/jdk/1.2/docs/guide/jni/spec/jniTOC.doc.html`).

Compilation en une librairie dynamique

Une fois l'implémentation des différentes méthodes effectuée, il faut les compiler. En fonction des systèmes d'exploitation et du compilateur, les commandes peuvent être différentes.

Sous Unix, avec le compilateur gcc

```
gcc -shared <les includes> <fichiers c> <librairie dynamique>
```

L'option shared stipule que l'on désire créer une libraire dynamique. <les includes> spécifie au moins deux chemins nécessaires pour le JNI, à savoir /include et /include/linux dans le répertoire d'installation de Java. Cette option est précédée du drapeau -I. <fichiers.c> contient l'implémentation de vos fonctions C. Enfin, <libraire dynamique> est le nom que l'on souhaite donner à la librairie. Cette option est précédée du drapeau -o.

Sous Windows, avec le compilateur cl

```
cl <les includes> -LD <fichiers c> <librairie dynamique>
```

Les options de compilation sont très proches de celles de gcc. Précédé de -I, <les includes> vaut \include et \include\win32, dans le répertoire d'installation de Java. -LD spécifie que l'on désire créer une librairie dynamique. <fichiers c> contient l'implémentation de vos fonctions C. <librairie dynamique> est le nom de la librairie que l'on désire créer.

Remarques à propos des librairies dynamiques Que ce soit sous Unix ou sous Windows, la nomenclature des librairies est définie. Ainsi, toute librairie dynamique sous Unix doit commencer par lib et se terminer avec l'extension .so. De même sous Windows, la librairie doit avoir comme extension .dll. De plus , il est important de savoir qu'une librairie n'est pas portable d'un système d'exploitation à un autre. Les librairies devront elles aussi être recompilées.

Exemple : Coucou

Ce petit programme permettra de mieux comprendre les différentes étapes sus-mentionnées. Ce programme prendra comme argument un mot et renverra tout simplement " coucou : <le mot> ". La partie Java imprimera sur l'écran " coucou : " et une fonction C se chargera d'afficher à la suite le mot passé en argument.

Nous allons donc dans un premier temps écrire le code Java principal.

```
public class Coucou {

    public native void affichC(String nom);

    static {
        System.loadLibrary("coucou");
    }

    public static void main(String[] argv){
        if (argv.length != 1){
            System.out.println("syntaxe : java Coucou <un_nom>");
```

```
            }
            else {
                  System.out.print("coucou : ") ;
                  new Coucou().affichC(argv[0]);
            }
      }
}
```

Cette classe sera sauvée dans un fichier que l'on nommera Coucou.java. Il faut alors compiler ce fichier avec *javac*.

```
javac Coucou.java
```

Dans un second temps, il faut récupérer la signature de la fonction à implémenter. Pour cela, il faut utiliser javah sur le fichier .class fraîchement compilé.

```
javah Coucou
```

Nous obtenons alors un fichier d'en-tête .h avec une fonction de signature

```
JNIEXPORT void JNICALL Java_Coucou_affichC (JNIEnv *, jobject, jstring);
```

Il faut à présent implémenter la fonction. Pour ce faire, on créera un fichier Coucou.c :

```
#include <stdio>
#include "Coucou.h"

JNIEXPORT void JNICALL Java_Coucou_affichC (JNIEnv *env , jobject obj,
jstring nom){
      const *jbyte str;
      str = (*env)->GetStringUTFChars(env,nom,NULL);
      printf("%s", str);
      (*env)->ReleaseStringUTFChars(env,nom,str);
      return ;
}
```

Deux choses sont intéressantes à noter. Tout d'abord, il ne faut pas oublier d'inclure le fichier généré par javah. Celui-ci contient la signature de la fonction et inclut *jni.h*, fourni par java (on le retrouve dans le répertoire \include de java). Le second point à observer est le double déférencement au niveau de la variable env, dû au fait que celle-ci est un pointeur qui pointe vers une table de pointeurs vers des fonctions.

On compile à présent le fichier coucou.c

- **Sous Unix :**
  ```
  gcc -shared -I/java/include -I/java/include/linux Coucou.c
  -o libcoucou.so
  ```
- **Sous Windows :**
  ```
  cl -Ic:\java\include  -Ic:\java\include\win32 -LD Coucou.c
  -Fecoucou.dll
  ```

Maintenant il suffit d'appeler le programme principal

```
java coucou Rudy
```

Ce qui aura pour effet de faire apparaître à l'écran : *Coucou : Rudy*

Problèmes

Quelques problèmes peuvent néanmoins apparaître.

- *javah Coucou : Class Coucou could not be found.* N'oubliez pas de préciser dans les variables d'environnement votre CLASSPATH
- *java Coucou Rudy : UnsatisfiedLinkError.* Au moins 2 causes peuvent être à l'origine de cette erreur.
  Soit java ne parvient pas à trouver la librairie créée. Pour cela, il faut ajuster la variable LD_LIBRARY_PATH au répertoire dans lequel se trouve votre librairie.

  Soit java ne parvient pas à trouver des méthodes définies dans la librairie. Ce problème peut venir de plusieurs choses. Il se peut tout simplement que les fonctions implémentées en C n'aient pas la même signature que les fonctions définies dans le fichier header .h. Une autre cause, moins triviale cette fois-ci, peut être due au fait que les fonctions natives sont utilisées au sein d'une hiérarchie en package. Pour remédier à cela, il convient de modifer légèrement la directive de génération du fichier .h. Supposons que nous ayons le fichier java déclarant les méthodes natives

suivantes :

```
package niveau1.niveau2.niveau3;

public class NativeMethod{

      static {
            System.loadLibrary("Test");
      }

      public native void method();

public static void main(String arg[]){
            new NativeMethod().method();
      }
}
```

Soit le directory de travail suivant : /develop. On définit les variables d'environnement suivantes :

```
CLASSPATH = /develop
LD_LIBRARY_PATH = /develop/niveau1/niveau2/niveau3
```

Il faut, pour générer correctement ce fichier d'en-tête, se placer dans le répertoire /develop et utiliser la commande suivante :
`javah -o NativeMethod.h -force niveau1.niveau2.niveau3.NativeMethod`
Pour le reste du développement, la marche à suivre est identique à ce qui a été dit dans les pages précédentes.

Annexe 2 : Tutorial d'utilisation de la plateforme

Prémisses

Ce programme est une application JAVA. Il est donc théoriquement censé tourner sous n'importe quel environnement. Néanmoins, certaines tâches ont été dévolues à des librairies C, interfacées dans JAVA grâce au JNI (*Java Native Interface*). Celui-ci permet de faire appel à des méthodes C (dites *natives*) depuis JAVA. A l'inverse, il est également possible de faire appel à des méthodes JAVA depuis un code C. Dans notre cas, des méthodes natives, sous forme de librairies dynamiques (.dll sous Windows ou .so sous Linux), ont été utilisées. Il faut donc disposer d'une version des librairies pour chaque environnement.

En outre, deux packages spéciaux ont été employés, à savoir zdicom.jar et emim.jar. Ils permettent de manier avec plus de simplicité les images au format DICOM supportées par la plate-forme.

Lancement du programme

Cette section décrit les paramètres nécessaires à la machine virtuelle afin de pouvoir utiliser correctement le logiciel.

Variables d'environnement

Pour pouvoir spécifier correctement les variables d'environnement nécessaires à la machine virtuelle, l'utilisateur doit savoir quelle commande permet de réaliser ces opérations. Cette commande système est *set* sous Windows, *export* sous Linux-Bash, *setenv* sous Linux-Csh. Il faut aussi noter que le séparateur de valeurs est le *;* sous Windows et *:* sous Linux.

La machine doit pouvoir trouver l'emplacement des classes compilées. Pour cela, on spécifiera l'emplacement du package Coreg.jar, contenant l'ensemble des classes compilées. De plus, comme la plate-forme utilise deux packages supplémentaires (cfr. *Prémisses*), il faut également le signaler à la machine virtuelle.

On spécifiera donc la variable CLASSPATH de la manière suivante :
- sous Windows

```
set CLASSPATH=<CheminDesClasses>\Coreg.jar;
```

```
<CheminDesCLasses>\emim.jar;<CheminDesCLasses>\DICOM.jar
```

- sous Linux-bash

```
export CLASSPATH=<CheminDesClasses>/Coreg.jar:
<CheminDesCLasses>/emim.jar:<CheminDesCLasses>/DICOM.jar
```

- sous Linux-csh

```
setenv CLASSPATH <CheminDesClasses>/Coreg.jar:
<CheminDesCLasses>/emim.jar:<CheminDesCLasses>/DICOM.jar
```

où <CheminDesClasses> représente le chemin d'accès aux classes.

Paramètres de la ligne de commande Java

Comme expliqué dans la première partie, la plate-forme de coregistration utilise des méthodes natives compilées dans des librairies C. La machine virtuelle Java doit savoir où ces librairies sont stockées sur le disque, afin de pouvoir y faire appel au moment désiré. Cela peut se faire via les paramètres de la machine virtuelle lors de l'exécution du programme. Pour ce faire, on appelera le flag *-D*, suivi du nom du paramètre correspondant à l'emplacement des méthodes natives, à savoir java.library.path suivi de l'emplacement des méthodes natives, à savoir Coreg/Native (si l'on se trouve sous Windows).

De plus, on devra veiller à ce que la machine virtuelle dispose d'assez de mémoire pour la manipulation des images, qui peuvent occuper, dans certains cas, énormement de place. Ainsi, si l'on ne souhaite pas se retrouver avec une erreur d'exécution de type *out of memory*, il sera nécessaire d'allouer une quantité non négligeable de mémoire, ce qui peut être fait via le flag *-Xmx* suivi d'un nombre représentant, en octets, la taille que l'on désire attribuer à la machine. 300.000.000 octets, ou quelque 300 Mo semble approprié.

Il faut bien évidemment préciser quelle classe on désire lancer. La classe principale, c'est-à-dire celle qui permet de lancer le programme, est la classe *CoregLauncher*, située dans le package Coreg.Couche4.

En résumé, la commande nécessaire pour lancer le programme est donc :

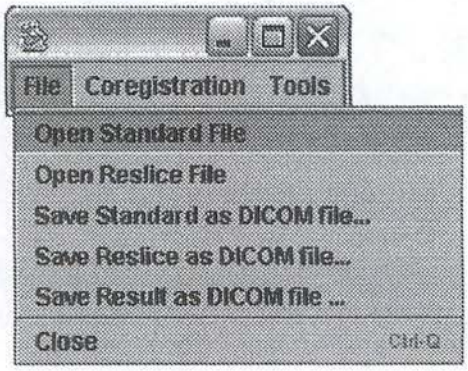
```
java -Djava.library.path=Coreg\Native
-Xmx300000000 Coreg.Couche4.CoregLauncher
```

Utilisation

Si toutes les variables et autres paramètres ont été correctement ajustés, l'interface principale devrait apparaître au bout de quelques instants, et devrait se présenter sour la forme suivante :

Ouverture de fichiers

Pour ouvrir un fichier, que ce soit un fichier standard ou à recaler, il suffit de cliquer dans le menu *File* et d'appuyer sur l'item désiré, à savoir *Open Standard File* ou *Open Reslice File.*

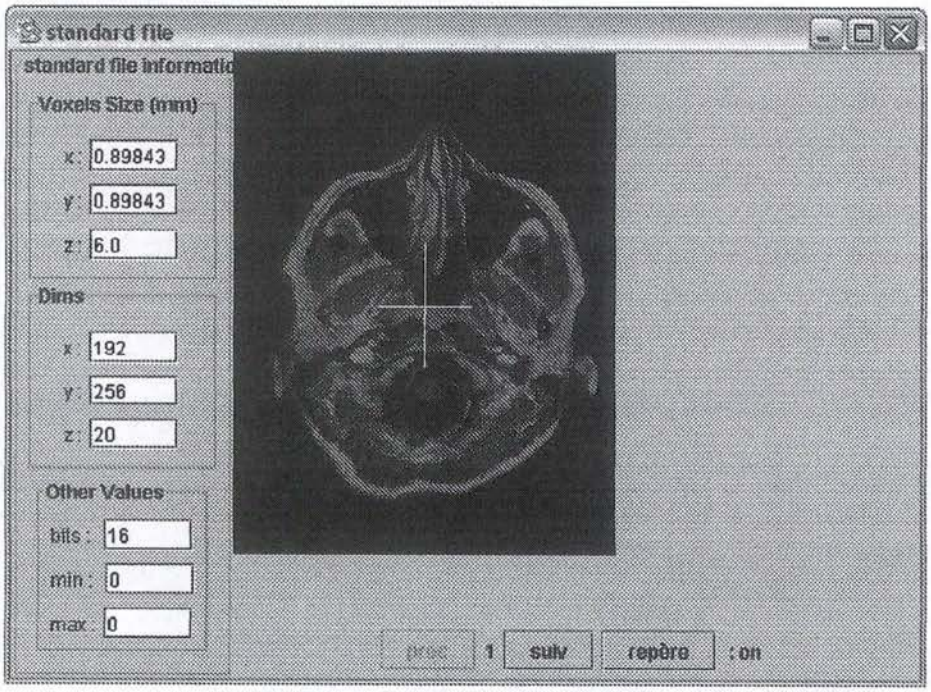


Une image médicale de format DICOM peut être codée de différentes manières. On distinguera ici deux grandes familles, selon qu'un fichier DICOM représente une image volumique ou qu'il ne représente qu'une seule slice. Dans ce dernier cas, un volume est défini par un ensemble de fichiers.

Si l'on souhaite ouvrir un fichier simple, il suffit de le sélectionner via la boîte de dialogue.

Dans le cas où l'on désire ouvrir une image médicale "multi-fichiers", il faut sélectionner l'ensemble des fichiers constituant le volume. Dans un tel cas de figure, il est plus aisé de regrouper l'ensemble des fichiers au sein d'un même répertoire. Il est à noter que dans

le cas où l'on ne sélectionne pas l'ensemble des fichiers nécessaires à la reconstitution du volume, un message d'erreur apparaît, signalant que des fichiers manquent ou sont corrompus. Si l'ouverture du (des) fichier(s) réussit, une nouvelle fenêtre apparaît, offrant un rendu de l'image ainsi que diverses informations la concernant.



Les informations fournies sont la taille des voxels (largeur, hauteur, profondeur) en millimètres, la résolution de l'image en nombre de voxels (largeur, hauteur, profondeur), la profondeur d'encodage des voxels (en bits/voxel) et les valeurs maximales et minimales de l'image.

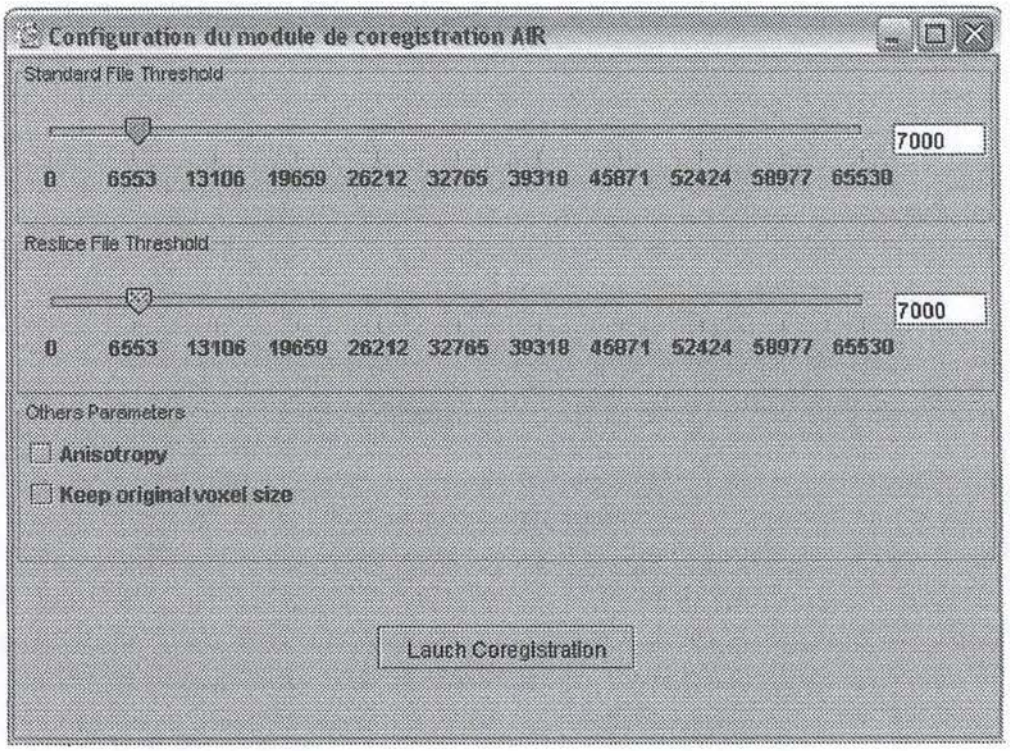
En outre, cette interface permet de se balader au sein des différentes tranches composant le volume. Pour cela, il suffit de cliquer sur les boutons *prec* et *suiv.*

Enfin, un marqueur est disponible pour offrir un indicateur visuel pour l'utilisateur, afin de marquer le centre de l'image.

Lancer une coregistration

Pour pouvoir lancer une coregistration d'images médicales, l'utilisateur doit bien évidemment ouvrir deux images médicales (cfr. section précédente). Une fois cette étape réalisée, il faut définir la stratégie que l'on va appliquer. Pour ce faire, il faut choisir dans

le menu *Coregistration* l'item *Choose Registration Strategy*. Une interface nous offre les différentes stratégies possibles, accompagnées d'une description textuelle de celle-ci. Une fois ce choix effectué, l'utilisateur va vouloir paramétrer le processus de coregistration. Il suffit pour cela de sélectionner, toujours dans le menu *Coregistration*, l'item *Start Registration*. Une nouvelle interface s'ouvrira, permettant de régler les différents paramètres du processus et ce, en fonction de la stratégie sélectionnée. L'image qui suit montre l'interface de paramétrage pour le module de coregistration AIR (le seul supporté par la plateforme jusqu'à présent).



Les deux slidebars situés en haut de la fenêtre permettent de régler le threshold du fichier standard et du fichier à recaler. Il s'agit du seuil à partir duquel AIR va considérer les voxels pour effectuer le recalage. *Anisotropy* est un prétraitement à sélectionner si la taille des voxels des fichiers standard et à recaler sont très différents. Enfin, *Keep Original Voxel Size* fait en sorte que le fichier résultat de la coregistration ait les mêmes tailles de voxel que le fichier original (par défaut, les voxels sont rendus cubiques).

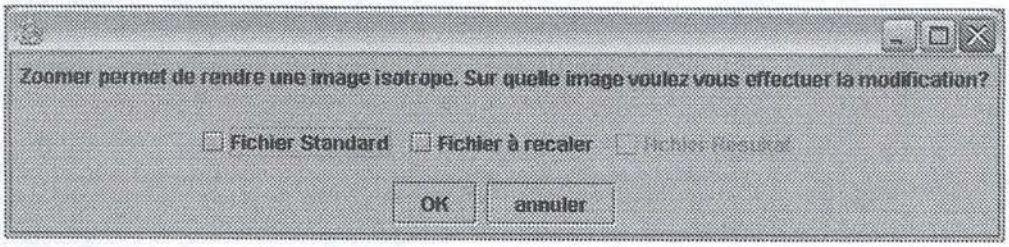
Une fois tout ceci paramétré, il suffit de lancer le processus de coregistration via le bouton prévu à cet effet. L'interface risque de ne plus répondre pendant quelques minutes, toutes les ressources étant attribuées à ce lourd processus.

Une fois la procédure terminée, une interface similaire à celle décrite à la section décrivant l'ouverture des fichiers apparaîtra, offrant un aperçu du résultat.

Tools

Le menu *Tools* offre un ensemble d'utilitaires, hors module de coregistration principal, permettant notamment d'effectuer des manipulations sur les images et de faire appel explicitement au Garbage Collector de Java. Cette dernière fonctionnalité n'est bien entendu pas un outil directement dédié à la coregistration.

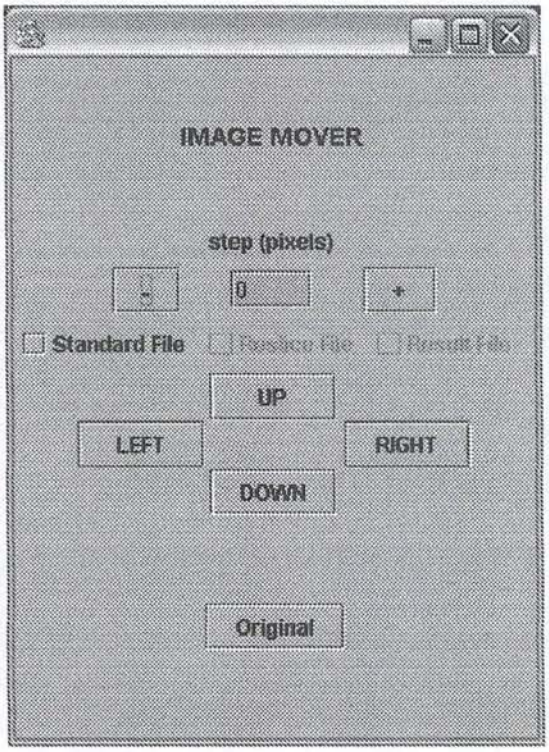
Zoomer Zoomer est un utilitaire permettant de rendre les voxels d'une image cubiques. Le moteur sous-jacent à cette transformation fait également partie du module AIR. Il va transformer l'image de telle sorte que chaque voxel ait comme dimension la plus petite dimension du voxel original. Pour utiliser cette fonctionnalité, il faut sélectionner l'item *Zoomer* du menu *Tools*. L'interface suivante apparaîtra.



Pour rendre une image isotrope, il suffit de sélectionner le fichier sur lequel on désire effectuer la modification, via les différents checkbox présents, et de cliquer sur OK. Après quelques instants, l'image réapparaîtra, modifiée en conséquence.

Mover Mover permet quant à lui d'effectuer des translations horizontales et verticales de l'image. On pourra par exemple translater l'image de 5 voxels vers la droite. Pratiquement, les voxels sortant du champ de définition de l'image se voient "détruits", alors que les 5 premiers voxels de chacunes des lignes sont mis à une valeur minimale, en fonction du type d'encodage des voxels (16 bits signés, non signés...). Pour utiliser cet outil, il suffit de cliquer sur l'item *Mover*, toujours dans le menu *Tools*. Une interface permettant de piloter les translations apparaîtra.

Son utilisation est assez aisée. Il faut sélectionner via les checkboxs l'image que l'on souhaite translater. Il faut bien entendu spécifier de combien de pas on désire translater



l'image. Cela peut être spécifé en cliquant sur les boutons "+" et "-". Il ne reste plus qu'à déplacer l'image dans la direction souhaitée. Si l'on veut déplacer l'image vers le haut, il suffit de cliquer sur *UP* et similairement pour les autres directions.

Enregistrement des fichiers

Il est bien évidemment intéressant de pouvoir sauver l'ensemble des modifications. Il est donc possible d'enregistrer les modifications apportées à chacune des images (standard, à recaler, résultat). L'utilisateur trouvera dans le menu *File* un item permettant de sauver chacune des ces images. L'utilisateur ne pourra néanmoins pas écraser un fichier par un autre.

Annexe C : Code de la plateforme

Les pages suivantes fourniront les sources d'une partie des classes de la plateforme. Voici le listing des classes dont le code est disponible ci-après.

Couche 1 : l'image médicale

- MedicalImage
- MedicalImage2D
- MedicalImageStack
- RawDataBuffer
- RawDataBufferByte

Les classes liées au contexte médical n'ont pas été inclues car elles ne sont pas intéressantes au niveau de leur structure et ne présentent pas d'algorithmes intéressants. Tous les types de buffers ne sont pas non plus inclus, étant donné leur similarité avec `RawDataBufferByte`.

Couche 2 : les éléments constitutifs

- Feature
- Voxel
- PlaneFeatureSet
- VoluminalFeatureSet
- PairingFunctional
- Variable
- BigSumFunction
- AIRPairingFunctional
- AIRPairingFunctional.c (code JNI)
- Tranformation

`OptimisationMethod` n'a pas été repris, n'ayant pas fait l'objet d'une implémentation. `Curve` et `ROI` n'ont pas été reprises, ces classes étant relativement similaires à `Voxel`. Enfin, seules quelques classes étendant `PairingFunctional` ont été présentées.

Couche 3 : la stratégie

- RegistrationStrategy

- AIRRegistrationStrategy
- RegistrationContext

Couche4 : l'éditeur

Aucune classe de cette couche n'a été inclue dans ces annexes. En effet, le nombre de lignes de code est relativement important et résulte principalement de la gestion des fenêtres. Il était donc peu intéressant de l'inclure.

Les outils auxiliaires

- Mover
- Zoomer
- ToAIR
- AIROpener
- CoregOpener
- DICOMDataObjectOperations

```java
package Coreg.Couche1;

/**
*MedicalImage est la classe abstraite à la base de toute modélisation d'une image médicale. Une image médicale peut être une image à deux dimensions (il s'agit d'une image plane, composée d'une seule slice) ou de dimensions supérieure. Une image plane sera représentée par la classe MedicalImage2D, dont les éléments constitutifs sont un contexte dans lequel l'image se définit (MedicalImageContext) ainsi que de données brutes, représentant la valeur des différents voxels. Pour une image à 3 dimensions (jusqu'à présent, aucune implémentation n'a pris en compte d'images de dimensions supérieures), on a recourt à la classe MedicalImageStack, composée d'un contexte ainsi que d'un ensemble de slices, chacune représentée par MedicalImage2D. On peut observer que le dénominateur commun entre MedicalImage2D et MedicalImageStack est le contexte de l'image. Cette élement est donc nécessaire à la classe MedicalImage qui est à la base de toute représentation d'image médicale dans le système.
*/

public class MedicalImage
{
    private MedicalImageContext context;

    /**
    *Crée une nouvelle instance de MedicalImage, en fournissant un contexte dans lequel se définit l'image
    *@param context le contexte
    */
    public MedicalImage(MedicalImageContext context){
        this.context = context;
    }

    /**
    *Renvoie le contexte de l'image médicale
    */
    public MedicalImageContext getMedicalImageContext(){
        return context;
    }

    /**
    *Spécifie le contexte dans lequel se définit l'image médicale
    */
    public void setMedicalImageContext(MedicalImageContext context){
        this.context = context;
    }
}

```

```java
package Coreg.Couche1;


/**
*MedicalImage2D permet la représentation d'images médicales à deux dimensions, autrement dit dotées d'une seule slice. Elle est définie par son contexte (comme tout image médicale) et par ses données brutes, c'est-à-dire la valeur des différents voxels de l'image
*/
public class MedicalImage2D extends MedicalImage{

    private RawDataBuffer data;

    /**
    *Crée une nouvelle instance de MedicalImage2D, en le dotant d'un contexte, et d'un buffer de données brutes
    *@param context le context
    *@param buffer le buffer de données brutes
    */
    public MedicalImage2D(MedicalImageContext context, RawDataBuffer buffer){
        super(context);
        this.data = buffer;
    }

    /**
    *Crée une nouvelle instance de MedicalImage2D, en le dotant d'un contexte
    */
    public MedicalImage2D(MedicalImageContext context){
        super(context);
    }

    /**
    *Spécifie le buffer de données brutes associés à l'image
    *@param buffer le buffer de données brutes
    */
    public void setRawDataBuffer(RawDataBuffer buffer){
        this.data = buffer;
    }

    /**
    *Renvoie le buffer de données brutes
    */
    public RawDataBuffer getRawDataBuffer(){
        return this.data;
    }


}
```

```java
package Coreg.Couche1;

/**
*MedicalImageStack représente une image médicale de dimension supérieure à deux. Les éléments constitutifs de cette classe sont tout d'abord un contexte, comme dans tout image médicale, et également un ensemble ordonné de MedicalImage, qui prendront pratiquement la forme d'une MedicalImage2D quand il s'agira de représenter des images en 3 dimensions, et de MedicalImageStack dans le cas d'images de dimensions supérieures à 3.
*/

public class MedicalImageStack extends MedicalImage{

    private MedicalImage[] stack;

    /**
    *Crée une nouvelle instance de MedicalImageStack, dont on spécifie le contexte, ainsi que le nombre de slices dont sera constitué l'image
    *@param context le contexte
    *@param numberOfSlices le nombre de slices dont est constitué l'image
    */

    public MedicalImageStack( MedicalImageContext context, int numberOfSlices){
        super(context);
        stack = new MedicalImage[numberOfSlices];
    }

    /**
    *Crée une nouvelle instance de MedicalImageStack en spécifiant le contexte
    *@param context le contexte
    */
    public MedicalImageStack (MedicalImageContext context){
        super(context);
    }

    /**
    *Spécifie le nombre de slices dont l'image sera constitué
    *@param numberOfSlices le nombre de slices
    */
    public void setNumberOfSlices(int numberOfSlices){
        stack = new MedicalImage[numberOfSlices];
    }

    /**
    *Renvoie le nombre de slices dont l'image est constitué
    */
    public int getNumberOfSlices(){
        return stack.length;
    }

    /**
    *Ajoute une MedicalImage à une position spécifiée
    *@param img l'image à ajouter
```

```
48      *@param pos la position à laquelle insérer l'image (la première image se situant à la position 0)
49      */
50      public void addMedicalImage(MedicalImage img, int pos){
51          stack[pos]=img;
52      }
53
54      /**
55      *Renvoie l'image située à une position particulière
56      *@param slice la position de l'image (à noter que la première image est à la position 0)
57      */
58      public MedicalImage getMedicalImage(int slice){
59          return stack[slice];
60      }
61
62      /**
63      *Renvoie l'ensemble des images constituant la MedicalImageStack
64      */
65      public MedicalImage[] getStack(){
66          return this.stack;
67      }
68  }
69
70
```

```java
package Coreg.Couche1;

import java.util.Vector;

/**
* RawDataBuffer est une classe abstraite permettant de représenter les données brutes des images, autrement dit les valeurs associées aux voxels. Cinq implémentations de cette classe abstraite ont été faites, chacune pour un type de donnée précis. On  dénotera les buffers de byte (RawDataBufferByte), de short (RawDataBufferShort), de unsigned short (RawDataBufferUShort), de integer (RawDataBufferInt) et de float(RawDataBufferFloat).
*<p>
*Chaque image en 2D possède un certain nombre de lignes et de colonnes. Une représentation simple de cette image est place chaque ligne de l'image l'une à la suite de l'autre, en formant un tableau dont le nombre d'éléments vaut le nombre de voxels par ligne, multiplié par le nombre de lignes (ou encore le nombre de colonnes multiplié par le nombre de lignes). Dans le cadre de RawDataBuffer, cette propriété n'apparait nulle part, mais dans les extensions de cette classe pour des types spécifiques, cette notion revient.
*/

public abstract class RawDataBuffer{

    private short dataType;
    /*
    Ce champs spécifie le type de grandeurs physiques, à savoir :
    byte = 1
    short = 2
    unsigned short = 3
    integer = 4
    float = 5
    */

    private short dataTypeSize[];

    /**
    *Crée une nouvelle instance de RawDataBuffer
    */
    public RawDataBuffer(){
        dataTypeSize = new short[5];
        dataTypeSize[0] = (short)8;
        dataTypeSize[1] = (short)16;
        dataTypeSize[2] = (short)16;
        dataTypeSize[3] = (short)32;
        dataTypeSize[4] = (short)32;
    }

    /**
    *Spécifie le type de donnée présente dans le buffer. Par convention, byte est représenté par 1, short par 2, unsigned short par 3, integer par 4 et float par 5
    *@param type le nombre représentant un type donné
    */

    public void setDataType(short type){
        this.dataType = type;
    }
```

```
45
46      /**
47      *Renvoie le type du buffer de données
48      */
49      public short getDataType(){
50          return this.dataType;
51      }
52
53      /**
54      *Renvoie le nombre de bits surlequel est encodé chaque valeur de voxel
55      */
56      public short getDataTypeSize(){
57          return this.dataTypeSize[dataType-1];
58      }
59  }
```

```java
package Coreg.Couche1;


/**
* RawDataBufferByte permet de représenter les buffers de données codées sous la forme de byte.
*/
public class RawDataBufferByte extends RawDataBuffer{

    private byte measure[];

    /**
    *Crée une nouvelle instance de RawDataBufferByte, en spécifiant le nombre d'éléments contenus dans le buffer de données
    *@param length le nombre d'éléments du buffer (autrement dit sa longueur)
    */
    public RawDataBufferByte(int length){
        super();
        this.measure = new byte[length];
        this.setDataType((short)1);

    }

    /**
    *Ajoute un élément au buffer à une certaine position
    *@param element l'élément à ajouter
    *@param pos la position à laquelle ajouter l'élément
    */
    public void addElement(byte element, int pos) {
        this.measure[pos]=element;

    }

    /**
    *Renvoie l'élément du buffer situé à une position spécifique
    *@param pos la position de l'élément que l'on souhaite obtenir
    */
    public byte getElement(int pos){
        return this.measure[pos];
    }

    /**
    *Renvoie l'ensemble des valeurs de pixels de l'image
    */
    public byte[] getValues(){
        return this.measure;
    }

}
```

```java
package Coreg.Couche2.Feature;

import java.util.Vector;

/**
 * Feature est une classe abstraite permettant une représentation de diverses caractéristiques intéressantes dans l'optique d'une coregistration d'images. On distingue trois familles de caractéristiques, dit de bas niveau (<i>Low Feature</i>) : le voxel, la courbe et la région d'intérêt. Ces caractéristiques peuvent non seulement s'exprimer sur une image plane, mais également dans un volume.
 *<p>
 * Chaque instance de la classe Feature contient un champs privé type qui décrit le type de caractéristique que l'on désire modéliser. Ainsi, si l'on désire spécifier comme caractéristique un ensemble bidimensionel de voxels, on créera une instance de la classe PlaneFeatureSet, dont on initialisera le champs type à "voxel".
 * @author Olivier Vaesen
 */
public abstract class Feature {

    private Vector featureSet;
    private String type;


    /**
    *Crée une nouvelle instance de Feature
    */
    public Feature(){
        featureSet = new Vector();
    }

    /**
    *Crée une nouvelle instance de Feature, en spéciant le quel est le type de caractéristique concerné.
    *@param type le type de caractéristique
    */
    public Feature(String type){
        featureSet = new Vector();
        this.type = type;
    }

    /**
    *Retourne le type de caractéristique concerné par l'instance de Feature
    */
    public String getType(){
        return type;
    }

    /**
    *Spécifie le type de caractéristique
    *@param type le type de caractéristique
    */
    public void setType(String type){
        this.type = type;
    }

```

```
48      /**
49      *Renvoie l'ensemble des caractéristiques
50      */
51
52      public Vector getFeatureSet(){
53          return featureSet;
54      }
55
56      /**
57      *Ajoute une caractéristique à l'ensemble des caractéristiques
58      *@param element la caractéristique à ajouter
59      *@exception WrongFeatureException est renvoyé si le type de l'élément n'est pas compatible avec le type spécifié dans Feature
60      */
61      public abstract void addFeature(Object element) throws WrongFeatureException;
62
63
64      /**
65      *Retire une caractéristique de l'ensemble des caractéristiques
66      *@param element la caractéristique à retirer
67      *@exception WrongFeatureException est renvoyé si le type de l'élément n'est pas compatible avec le type spécifié dans Feature
68      */
69      public abstract void removeFeature(Object element) throws WrongFeatureException;
70  }
71
```

```java
package Coreg.Couche2.Feature;

/**
 * Voxel permet la représentation de caractéristique de coregistration de type voxel, c'est-à-dire un point possédant une position spatiale, représentée par trois valeurs, ainsi que d'une valeur physique associée à ce voxel.
 */
public class Voxel {

    private int width;
    private int height;
    private int depth;
    private double value;


    /**
    *Crée une nouvelle instance de Voxel, avec une position et une valeur définie
    *@param width la position en abscisse
    *@param height la position en ordonnée
    *@param depth la postion en profondeur
    *@param value sa valeur physique
    */

    public Voxel(int width, int height, int depth, double value){
        this.width = width;
        this.height = height;
        this.depth = depth;
        this.value = value;
    }

    /**
    *Retourne la position en abscisse du voxel
    */

    public int getWidth(){
        return width;
    }

    /**
    *Retourne la position en ordonnée du voxel
    */
    public int getHeight(){
        return height;
    }

    /**
    *Retourne la position en profondeur du voxel
    */
    public int getDepth(){
        return depth;
    }

```

```
51      /**
52      *Retourne la valeur physique du voxel
53      */
54      public double getValue(){
55          return value;
56      }
57
58      /**
59      *Teste si deux voxel sont identiques, c'est-à-dire s'ils possèdent la même position spatiale (on ne prête pas attention à la valeur
        du voxel)
60      *@param element le voxel avec lequel on souhaite comparer l'instance courante de Voxel
61      *@exception WrongFeatureException si l'element n'est pas une instance de Voxel
62      */
63      public boolean isEqual(Voxel element) throws WrongFeatureException{
64          if (element.getClass().getName().compareTo("Voxel")==0){
65              return ((((this.width==element.getWidth())&&(this.height==element.getHeight()))))&&(this.depth==element.getDepth()));
66          }
67          else {
68              throw new WrongFeatureException();
69          }
70      }
71
72
73      /**
74      *Spécifie la position en abscisse du voxel
75      *@param width sa position en abscisse
76      */
77      public void setWidth(int width){
78          this.width = width;
79      }
80
81      /**
82      *Spécifie la position en ordonnée du voxel
83      *@param height sa position en ordonnée
84      */
85      public void setHeight(int height){
86          this.height = height;
87      }
88
89      /**
90      *Spécifie la position en profondeur du voxel
91      *@param depth sa position en profondeur
92      */
93      public void setDepth(int depth){
94          this.depth = depth;
95      }
96
97      /**
98      *Spécifie la valeur associée au voxel
99      *@param value la valeur physique associée au voxel
100     */
```

```
101     public void setValue(double value){
102         this.value = value;
103     }
104 }
105
```

```java
package Coreg.Couche2.Feature;

/**
 * PlaneFeatureSet permet la représentation d'ensemble de caractéristiques. PlaneFeatureSet représentera par exemple un ensemble de
 Voxel. Il est important de noter qu'une instance de PlaneFeatureSet ne pourra pas contenir différents types de caractéristiques de
 base, telles que Voxel et Curve.
 */
public class PlaneFeatureSet extends Feature{

    /*le champ type permet de spécifier explicitement le type de PlaneFeatureSet
     * auquel on a à faire. On supposera que ce champ peut prendre trois valeurs,
     * à savoir "curve", "voxel" et "ROI". L'interêt de ce champ est de ne pas
     * permettre d'insérer différents type d'éléments dans un même vecteur.*/

     /**
     *Crée une nouvelle instance de PlaneFeatureSet en spécifant le type des caractéristiques qu'il regroupera
     *@param type le type des caractéristiques concernées (par exemple "voxel", "curve",...)
     */

    public PlaneFeatureSet(String type){
        super(type);
    }



    public void addFeature(Object element)throws WrongFeatureException{
        if (element.getClass().getName().compareTo(this.getType())==0){
            this.getFeatureSet().addElement(element);
        }
        else{
            throw new WrongFeatureException();
        }
    }

    public void removeFeature(Object element)throws WrongFeatureException{
        if (element.getClass().getName().compareTo(this.getType())==0){
            this.getFeatureSet().removeElement(element);
        }
        else{
            throw new WrongFeatureException();
        }
    }
}

```

```java
package Coreg.Couche2.Feature;

/**
 *VoluminalFeatureSet permet de représenter en ensemble de caractéristique planes, elles-même définies  grâce à PlaneFeatureSet.
 */
public class VoluminalFeatureSet extends Feature{


    /**
    *Crée une nouvelle instance de VoluminalFeatureSet
    *@param type le tye des caractéristiques de base
    */
    public VoluminalFeatureSet(String type){
        super(type);
    }

    public void addFeature(Object element)throws WrongFeatureException{
        try {
            PlaneFeatureSet planeFeatureSet = (PlaneFeatureSet)element;
            if (planeFeatureSet.getType().compareTo(this.getType())==0){
                this.getFeatureSet().addElement(element);
            }
            else {
                throw new WrongFeatureException();
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }

    public void removeFeature(Object element) throws WrongFeatureException{
        try{
            PlaneFeatureSet planeFeatureSet = (PlaneFeatureSet)element;
            if (planeFeatureSet.getType().compareTo(this.getType())==0){
                this.getFeatureSet().addElement(element);
            }
            else {
                throw new WrongFeatureException();
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}

```

```java
package Coreg.Couche2.PairingFunctional;

/**
*PairingFunctional est une classe abstraite permettant de représenter les fonctions de mesure de la distance entre divers images. Deux méthodes ont été définies, permettant d'assigner des valeurs aux variables de la fonction et, une fois cela fait, de calculer la valeur de la fonction.
*<p>
*Pratiquement, cette représentation n'a pas encore été employée, car le seul alogrithme jusqu'ici utilisé pour mesurer la distance entre deux images est AIR, codé en C. On a fait appel au JNI pour l'incorporer dans la plateforme java. Les diverses fonctions définies dans ce package n'ont encore jamais été utilisées, exceptés AirPairingFunctional et PairingFunctional.
*/
public abstract class PairingFunctional {

    /**
    *Permet de calculer la valeur de la fonction définie représentée par l'instance de PairingFunctional
    *@exception NoAttributedValueException est renvoyé si une des variables de la fonction n'a pas de valeur lors du calcul de la fonction
    */
    public abstract double execute() throws NoAttributedValueException;

    /**
    *Permet d'assigner une valeur à une des variables de la fonction représentée par l'instance de PairingFunctional
    *@param variable le nom de la variable
    *@param value la valeur à assigner à la variable
    */
    public abstract void assignValue(String variable, double value);
}

```

```java
package Coreg.Couche2.PairingFunctional;

/**
 * Variable permet de représenter les variables d'une fonction. On peut définir deux états d'une variable. Soit la variable est non initialisée, dans ce cas, tout tentative de calcul de la fonction dans laquelle cette variable intervient, renverra une exception de type NoAttributedValueException. Si elle est assignée, la variable renverra la valeur à laquelle elle a été initialisée
 */
public class Variable extends PairingFunctional{

    private String variable;
    private boolean assigned;
    private double value;

    /**
    *Crée une nouvelle instance de Variable
    *@param variable le nom de la variable
    */
    public Variable(String variable){
        this.variable = variable;
        this.assigned = false;
    }

    public double execute() throws NoAttributedValueException{
        if (assigned){
            return value;
        }
        else{
            throw new NoAttributedValueException(variable);
        }
    }

    public void assignValue(String variable, double value){
        if (variable.compareTo(this.variable)==0){
            assigned = true;
            this.value = value;
        }
    }
}

```

```java
package Coreg.Couche2.PairingFunctional;

/**
*BigSumFunction permet de représenter et de calculer la fonction <i>grande somme</i>.
*/
public class BigSumFunction extends PairingFunctional{
    private PairingFunctional func;
    private String variable;
    private int minValue;
    private int maxValue;

    /**
    *Crée une nouvelle instance de BigSumFunction
    *@param func la fonction de laquelle on désire effectuer la somme
    *@param variable le nom de l'indice
    *@param minValue la valeur minimale de l'indice
    *@param maxValue la valeur maximale de l'indice
    */
    public BigSumFunction (PairingFunctional func, String variable, int minValue, int maxValue){
        this.func = func;
        this.variable = variable;
        this.minValue = minValue;
        this.maxValue = maxValue;
    }

    public double execute() throws NoAttributedValueException{
        try{
            double result = 0;
            for (int i = minValue; i<=maxValue; i++){
                func.assignValue(variable,i);
                result = result + func.execute();
            }
            return result;
        }
        catch(NoAttributedValueException e){
            throw e;
        }
    }

    public void assignValue(String variable, double value){
        func.assignValue(variable,value);
    }
}

```

```java
package Coreg.Couche2.PairingFunctional;

import Coreg.Couche2.Transformation.Transformation;
import Coreg.Native.NativeException;

/**
*AirPairingFunctional est une classe un peu particulière, car les méthodes employées à l'intérieur de cette classe sont définies à l'aide de méthode natives. Ces méthodes natives ont été sélectionnées au sein du module de coregistration d'image médicales AIR (Automated Image Registration). Il s'agit d'algorithmes écrits en code C. L'algorithme utilisé contient non seulement des méthodes destinés à l'appariement fonctionnel, mais également des méthodes d'extraction de caractéritiques et d'optimisation. Le code fourni étant assez complexe et relativement peu documenté, le module de coregistration AIR intégré dans sa globalité dans cette classe. C'est pour cela qu'on y trouve un champs de type Transformation (qui reprend la matrice des coefficients de la transformation) et un champ destiné aux paramètres
*/
public class AirPairingFunctional extends PairingFunctional{

    /**
    *La matrice de transformation, c'est-à-dire le résultat de la recherche de coefficients afin que les images soient le plus semblable possible.
    */
    public Transformation matrice;

    /**
    *Les paramètres passés en argument pour piloter le module de coregistration AIR
    */
    public String[] params;


    /**
    *Crée une nouvelle instance de AirPairingFunctional
    */
    public AirPairingFunctional(){
        matrice = new Transformation();
    }

    /*
    Charge la librairie dynamique AIR
    */
    static{
        System.loadLibrary("AIR");
    }

    public double execute() throws NoAttributedValueException{
        try
        {
            alignLinear(params.length,params);
            return 1;
        }
        catch (Exception e)
        {
            return 0;
```

```
45          }
46      }
47
48
49      public void assignValue(String variable, double value){
50      }
51
52
53
54      private void alignLinear(int length, String[] param) throws NativeException{
55          try
56          {
57              this.doAlignLinear(length,param);
58          }
59          catch (Exception e)
60          {
61              e.printStackTrace();
62              return;
63          }
64
65          double[][] broll = getNativeMatrice();
66
67          if (broll==null)
68          {
69              System.out.println("Erreur : aucune transformation trouvée");
70          }
71          else matrice.setMatrice(broll);
72      }
73
74      public String[] getParams(){
75          return this.params;
76      }
77
78      private native void doAlignLinear(int length, String[] param) throws NativeException;
79
80      private native double[][] getNativeMatrice() throws NativeException; //se trouve dans le fichier de params.air
81
82  }
```

```c
#include <stdio.h>
#include "alignlinear.c"
#include "AirPairingFunctional.h"

JNIEXPORT void JNICALL Java_Coreg_Couche2_PairingFunctional_AirPairingFunctional_doAlignLinear(JNIEnv * env, jobject obj, jint length,
jobjectArray params){

    int i;
    jstring el;
    char* tab[256];

    //on supposera que l'on n'aura jamais de commandes avec plus de 256 arguments.

    tab[0] = "main_alignlinear";

    for (i = 0; i < length ; i++)
    {
        el = (jstring)(*env)->GetObjectArrayElement(env,params,i);
        tab[i+1] = (*env)->GetStringUTFChars(env, el, NULL);
    }

    i = main_alignlinear(length+1,tab); //on appelle la méthode AIR alignlinear
    if (i != 0)
    {
        (*env)->ThrowNew(env, (*env)->FindClass(env,"Coreg/Native/NativeException"),"Mauvais paramètres");
    }
}


JNIEXPORT jobjectArray JNICALL Java_Coreg_Couche2_PairingFunctional_AirPairingFunctional_getNativeMatrice(JNIEnv * env, jobject obj){
    FILE *fp = fopen("param","rb");// on a supposé que le fichier contenant la transformation se nommait param
    struct AIR_Air16 coreg;

    int i;
    int j;
    jdouble *doubleArrayElements;
    jdoubleArray tmpArray;
    jobjectArray outerArray;

    if (!fp)
    {
        printf("impossible d'ouvrir le fichier param\n");
        return NULL;
    }

    if (fread(&coreg,1,sizeof(struct AIR_Air16),fp)!=sizeof(struct AIR_Air16))
    {
        printf("Erreur de lecture du fichier : mauvais format attendu\n");
        return NULL;
    }

```

```
51      outerArray = (*env)->NewObjectArray(env,4,(*env)->FindClass(env,"java/lang/Object"),NULL);
52
53      for (i=0;i<(*env)->GetArrayLength(env,outerArray);i++)
54      {
55          tmpArray = (*env)->NewDoubleArray(env,4);
56          doubleArrayElements = (jdouble*)(*env)->GetDoubleArrayElements(env,tmpArray,NULL);
57          for (j=0;j<(*env)->GetArrayLength(env,tmpArray);j++)
58          {
59              printf("%lf\n",coreg.e[j][i]);
60              doubleArrayElements[j]=coreg.e[j][i];
61          }
62          printf("\n");
63          (*env)->ReleaseDoubleArrayElements(env,tmpArray,doubleArrayElements,0);
64          (*env)->SetObjectArrayElement(env,outerArray,i,tmpArray);
65
66      }
67      fclose(fp);
68      return outerArray;
69  }
70
71
```

```java
package Coreg.Couche2.Transformation;

/**
 * Transformation représente une matrice de transformation, on applique cette matrice sur l'image afin de rendre celle-ci "similaire" à
 une autre
 */
public class Transformation {

    private double paramTab[][];

    /**
    * Crée une nouvelle instance de Transformation
    */
    public Transformation(){

        paramTab = new double[4][4];
    }

    /*Convention de représentation pour la matrice de transformation
     *
     *                  |a00 a01 a02 a03|
     *                  |a10 a11 a12 a13|
     *       paramTab = |a20 a21 a22 a23|
     *                  |a30 a31 a32 a33|
     */

    /**
    *Spécifie la valeur d'un des élements de la matrice
    *@param element la valeur à placer dans la matrice
    *@param line la position en ordonnée de l'élément à insérer
    *@param row la position en abscisse de l'élément à insérer
    */
    public void setElementAt(double element, int line, int row){
        paramTab[line][row] = element;
    }

    /**
    *Renvoie un élement spécifique de la matrice
    *@param line la position en ordonnée de l'élément
    *@param row la position en abscisse de l'élément
    */

    public double getElementAt(int line, int row){
        return paramTab[line][row];
    }

    /**
    *Spécifie les valeurs de la matrice
    *@param tab la matrice des éléments
    */
    public void setMatrice(double[][] tab){
```

```
51              paramTab = tab;
52          }
53
54          /**
55          *Retourne la matrice des paramètres de transformation
56          */
57
58          public double[][] getMatrice(){
59              return paramTab;
60          }
61      }
62
```

```java
package Coreg.Couche3.RegistrationStrategy;

import Coreg.Couche2.Feature.*;
import Coreg.Couche2.Transformation.*;
import Coreg.Couche2.PairingFunctional.*;

import Coreg.Couche2.OptimisationMethod.*;
import Coreg.Couche1.*;


/**
*RegistrationStrategy définit l'ensemble des méthodes (abstraites) communes à toute stratégie de coregistration.
*/
public abstract class RegistrationStrategy
{
    private Feature img1Features;
    private Feature img2Features;
    private Transformation trans;
    private PairingFunctional pf;
    private OptimisationMethod mo;
    private MedicalImage imgResult;
    private String[] params;


    /**
    *Renvoie la matrice des coefficients de transformation à appliquer à l'image à recaler.
    */
    public Transformation getTransformation(){
        return trans;
    }

    /**
    *Permet de spécifier la metrice de transformation
    */
    public void setTransformation(Transformation trans){
        this.trans = trans;
    }

    /**
    *Permet de lancer de le processus de coregistration
    *@param img1 l'image modèle
    *@param img2 l'image à recaler
    *@param pf la fonctionelle d'appariement
    *@param om la méthode d'optimisation
    */
    public abstract void doRegistration(MedicalImage img1, MedicalImage img2, PairingFunctional pf, OptimisationMethod om);

    /**
    *Permet la visualisation du résultat par rapport au modèle. Il faut voir si cette méthode a effectivement place au sein des
    stratégies, plutôt que de regrouper les différentes méthodes de visualisation de résultat au sein d'un package dans la plateforme.
    *@param img l'image modèle
```

```
51      *@param imgResult le résultat de la coregistration
52      */
53      public abstract void visualizeResult(MedicalImage img, MedicalImage imgResult);
54
55      /**
56      *Permet d'effectuer la fusion entre l'image à recaler et l'image effectivement recalée. Encore une fois, je ne sais pas si cette
        méthode a effectivement sa place au sein d'une stratégie.
57      */
58      public abstract void doFusion(MedicalImage img, MedicalImage imgResult);
59
60
61      /**
62      *Permet de trouver la matrice de transformation, à partir des caractéristiques issues des deux images, grâce à la fontionelle
        d'appariement et d'une méthode d'optimisation
63      *@param img1Feature les caractéristiques issues de l'image modèle
64      *@param img2Feature les caractéristiques issues de l'image à recaler
65      *@param f la fonctionelle d'appariement
66      *@param o la méthode d'optimisation
67      */
68      public abstract Transformation researchTransformation(Feature img1Feature, Feature img2Feature, PairingFunctional f,
        OptimisationMethod o);
69
70      /**
71      *Permet d'appliquer la matrice de transformation sur l'image à recaler
72      *@param img l'image surlaquelle on souhaite appliquer la matrice de transformation.
73      *@param t la matrice de transformation
74      */
75      public abstract MedicalImage applyTransformationOnSecondImage(MedicalImage img, Transformation t);
76
77      /**
78      *Permet d'extraire un ensemble de caractéristiques à partir d'une image
79      *@param img l'image de laquelle on souhaite extraire l'ensemble de caractéristiques
80      */
81      public abstract Feature extractFeatures(MedicalImage img);
82
83      /**
84      *
85      */
86      public MedicalImage getImgResult(){
87          return imgResult;
88      }
89
90      public void setImgResult(MedicalImage img){
91          this.imgResult = img;
92      }
93
94      public void setParams(String[] params){
95          this.params = params;
96      }
97
98      public String[] getParams(){
```

```
 99          return this.params;
100      }
101  }
```

```java
package Coreg.Couche3.RegistrationStrategy;

import Coreg.Couche2.Transformation.*;
import Coreg.Couche2.PairingFunctional.*;
import Coreg.Couche2.OptimisationMethod.*;
import Coreg.Couche2.Feature.*;
import Coreg.Couche1.*;
import Coreg.Utils.*;
import Coreg.Native.NativeException;

import java.io.File;


//nb sur les parametres reçus. si (params[params.length-1] == "cubic"), il faut ajouter aux parametres du reslice l'option -k afin de rendre les voxels cubics

public class AIRRegistrationStrategy extends RegistrationStrategy{

    static{
        System.loadLibrary("Reslice");
    }

    private boolean cubic = false;

    public Transformation researchTransformation(Feature img1, Feature img2, PairingFunctional f, OptimisationMethod m){
        AirPairingFunctional g = (AirPairingFunctional)f;
        g.params = this.deriveAlignParamsFromParams();
        for (int i = 0; i<g.params.length ;i++ )
        {
            System.out.println(g.params[i]);
        }

        try
        {
            g.execute();
            return g.matrice;
        }
        catch (Exception e)
        {
            e.printStackTrace();
            return null;
        }

    }


    public Feature extractFeatures(MedicalImage img){
        return null;
    }

    /*le fichier de paramètres pour le reslice est généré à partir du fichier de paramètres de alignlinear */
```

```
51
52      public MedicalImage applyTransformationOnSecondImage(MedicalImage img, Transformation t){
53          String[] tmp = deriveResliceParamsFromAlignParams();
54          System.out.println(tmp);
55          doReslice(tmp.length,tmp);
56          // dans un premier temps, le résultat du doReslice sera deux fichiers .hdr et .img sur le disque dur. On se chargera par le
            suite de placer cette structure en mémoire.
57          AIROpener ao = new AIROpener();
58          return ao.openImage(tmp[1]);
59      }
60
61      public String[] deriveAlignParamsFromParams(){
62          String[] temp = this.getParams();
63          String[] res;
64          if (temp[(temp.length - 1)].compareTo("cubic")==0)
65          {
66              cubic = true;
67              res = new String[temp.length-1];
68              for (int i = 0; i < res.length ; i++)
69              {
70                  res[i] = temp[i];
71              }
72          }
73          else {
74              res = temp;
75          }
76          commandDebug(res);
77          return res;
78      }
79
80      public String[] deriveResliceParamsFromAlignParams(){
81
82          String[] res;
83
84          if (cubic)
85          {
86              res = new String[3];
87              res[0] = this.getParams()[2];
88              res[1] = new String("temp"); // fichier temporaire contenant les rawdata et le header
89              res[2] = new String("-k");
90              return res;
91          }
92
93          res = new String[2];
94          res[0] = this.getParams()[2];
95          res[1] = new String("temp");
96          commandDebug(res);
97          return res;
98      }
99
100     private void commandDebug(String[] com){
```

```
101         for (int i=0; i<com.length ; i++)
102         {
103             System.out.println(com[i]);
104         }
105     }
106
107     /*la méthode considère que les paramètres sont syntaxiquement corrects*/
108     public native void doReslice(int length, String[] params);
109
110     public void doRegistration(MedicalImage img1, MedicalImage img2, PairingFunctional f, OptimisationMethod o){
111
112         try
113         {
114             ToAIR to1 = new ToAIR(img1,this.getParams()[0]);
115             to1.convertToAIRFile();
116             Feature img1Feature = null;
117
118             ToAIR to2 = new ToAIR(img2,this.getParams()[1]);
119             to2.convertToAIRFile();
120             Feature img2Feature = null;
121
122             Transformation trans = this.researchTransformation(img1Feature, img2Feature, f, null);
123             setTransformation(trans);
124             MedicalImage imgRes = applyTransformationOnSecondImage(img2,trans);
125             this.setImgResult(imgRes);
126
127             deleteFile("temp.hdr");
128             deleteFile("temp.img");
129             deleteFile("reslice.hdr");
130             deleteFile("reslice.img");
131             deleteFile("modele.hdr");
132             deleteFile("modele.img");
133             deleteFile("param");
134         }
135         catch (Exception e)
136         {
137             System.out.println("y a eu exception");
138         }
139
140
141     }
142
143     private void deleteFile(String fileName){
144         File file = new File(fileName);
145         if (file.exists())
146         {
147             if (file.delete())
148             {
149                 System.out.println(file.getPath()+" a été correctement effacé");
150             }
151             else System.out.println("Erreur lors de l'effacement du fichier");
```

```
152
153             }
154             else System.out.println("Erreur lors de l'effacement du fichier");
155         }
156
157         public void visualizeResult(MedicalImage img, MedicalImage imgResult){
158
159         }
160
161         public void doFusion(MedicalImage img, MedicalImage imgResult){
162         }
163     }
164
```

```java
package Coreg.Couche3;

import Coreg.Couche2.PairingFunctional.*;
import Coreg.Couche2.OptimisationMethod.*;
import Coreg.Couche1.*;
import Coreg.Couche3.RegistrationStrategy.*;


/**
*RegistrationContext est une classe définissant les stratégies employées afin de procéder à une coregistration. pour ce faire, cette classe nécessite une stratégie de coregistration, dont la classe RegistrationStrategy assure la définition de l'ensemble des méthodes communes à toute stratégie de coregistration. Elle nécessite bien évidemment les deux images que l'on souhaite coregistrer. Enfin, elle requiert une fonctionelle d'appariement et une méthode d'optimisation
*/
public class RegistrationContext
{
    private RegistrationStrategy strategy;
    private MedicalImage img1;
    private MedicalImage img2;
    private PairingFunctional pf;
    private OptimisationMethod om;
//  private String transfotype;


    /**
    *Crée une nouvelle instance de RegistrationContext
    *@param strategy la stratégie de coregistration
    *@param std_img l'image qui servira de modèle à la coregistration
    *@param rsl_img l'image qui sera recalée
    *@param pf la fonctionelle d'appariement
    *@param om la méthode d'optimisation
    */
    public RegistrationContext(RegistrationStrategy strategy, MedicalImage std_img, MedicalImage rsl_img, PairingFunctional pf, OptimisationMethod om){
        this.strategy = strategy;
        this.img1 = std_img;
        this.img2 = rsl_img;
        this.pf = pf;
        this.om = om;
    }

    /**
    *Lance la procédure de recalage des deux images
    */
    public void executeRegistration() {
        strategy.doRegistration(img1, img2, pf, om);
//      strategy.visualizeResult(img1, strategy.getImgResult());
//      strategy.doFusion(img1,strategy.getImgResult());
    }

    /**
```

```
48      *Passe des paramètres à la stratégie de coregistration
49      *@param params le tableau de paramètres
50      */
51      public void setParams(String[] params){
52          this.strategy.setParams(params);
53      }
54
55      /**
56      *Renvoie les paramètres affectés à la stratégie de coregistration
57      */
58      public String[] getParams(){
59          return strategy.getParams();
60      }
61
62      /**
63      *Renvoie l'image résultant de l'exécution de la stratégie de coregistration
64      */
65      public MedicalImage getImgResult(){
66          return strategy.getImgResult();
67      }
68  }
```

```
48      *Passe des paramètres à la stratégie de coregistration
49      *@param params le tableau de paramètres
50      */
51      public void setParams(String[] params){
52          this.strategy.setParams(params);
53      }
54
55      /**
56      *Renvoie les paramètres affectés à la stratégie de coregistration
57      */
58      public String[] getParams(){
59          return strategy.getParams();
60      }
61
62      /**
63      *Renvoie l'image résultant de l'exécution de la stratégie de coregistration
64      */
65      public MedicalImage getImgResult(){
66          return strategy.getImgResult();
67      }
68  }
```

```java
package Coreg.Tools;

import Coreg.Couche1.*;
import Coreg.Utils.Tools;

public class Mover
{
    public final static int LEFT = 1;
    public final static int RIGHT = 2;
    public final static int UP = 3;
    public final static int DOWN = 4;

    private MedicalImage img;
    private MedicalImage copy;

    private BasicImageInfo bii;

    public Mover(MedicalImage image){
        if (image == null)
        {
            return;
        }
        this.img = image;
        this.bii = this.img.getMedicalImageContext().getGeneralInfo().getBasicImageInfo();
        if (bii==null)
        {
            System.out.println(bii.getX_dim());
        }

    }

    public MedicalImage move(int key, short step){

        switch (key)
        {
        case LEFT:
        case RIGHT:
            if (step>=bii.getX_dim())
            {
                Tools.errorMessage("Pas trop grand par rapport à la taille de l'image");
                step = 0;
            }
            break;
        case UP:
        case DOWN:
            if (step>=bii.getY_dim())
            {
                Tools.errorMessage("Pas trop grand par rapport à la taille de l'image");
                step = 0;
            }
            break;
```

```
52          default:
53              return null;
54          }
55
56          if (img.getClass().getName().compareTo("Coreg.Couche1.MedicalImage2D")==0)
57          {
58              switch (key)
59              {
60              case LEFT:
61                  return moveLeft((MedicalImage2D)img,step);
62              case RIGHT:
63                  return moveRight((MedicalImage2D)img,step);
64              case UP:
65                  return moveUp((MedicalImage2D)img,step);
66              case DOWN:
67                  return moveDown((MedicalImage2D)img,step);
68              default:
69                  return null;
70              }
71          }
72          else{ // MedicalImageStack
73
74              MedicalImage tmp;
75              for (int i=0; i<bii.getZ_dim() ;i++ )
76              {
77                  MedicalImage2D slice = (MedicalImage2D)((MedicalImageStack)img).getMedicalImage(i);
78                  switch (key)
79                  {
80                  case LEFT:
81                      tmp = moveLeft(slice,step);
82                      break;
83                  case RIGHT:
84                      tmp = moveRight(slice,step);
85                      break;
86                  case UP:
87                      tmp = moveUp(slice,step);
88                      break;
89                  case DOWN:
90                      tmp = moveDown(slice,step);
91                      break;
92                  default :
93                      return null;
94                  }
95                  ((MedicalImageStack)img).addMedicalImage(tmp,i);
96              }
97              return img;
98          }
99      }
100
101
102     private MedicalImage2D moveRight(MedicalImage2D slice, short step){
```

```
103
104         short dataType = slice.getRawDataBuffer().getDataType();
105         int offset;
106         for (int k=0; k<bii.getY_dim(); k++)
107         {
108             offset = k*bii.getX_dim();
109             switch (dataType)
110             {
111             case 2:
112                 RawDataBufferShort rdbS = (RawDataBufferShort)slice.getRawDataBuffer();
113                 for (int j=bii.getX_dim() - 1, i=bii.getX_dim() - (step + 1); i>=0; i--,j--)
114                 {
115                     rdbS.addElement(rdbS.getElement(offset+i),offset+j);
116                 }
117                 for (int i=0; i<step ; i++ )
118                 {
119                     rdbS.addElement(Short.MIN_VALUE,offset+i);
120                 }
121                 break;
122
123             case 3:
124                 RawDataBufferUShort rdbU = (RawDataBufferUShort)slice.getRawDataBuffer();
125                 for (int j=bii.getX_dim() - 1, i=bii.getX_dim() - (step + 1); i>=0; i--,j--)
126                 {
127                     rdbU.addElement(rdbU.getElement(offset+i),offset+j);
128                 }
129                 for (int i=0; i<step ; i++ )
130                 {
131                     rdbU.addElement(Short.MIN_VALUE,offset+i);
132                 }
133                 break;
134             default :
135                 break;
136             }
137         }
138         return slice;
139     }
140
141
142
143     private MedicalImage2D moveLeft(MedicalImage2D slice, short step){
144         short dataType = slice.getRawDataBuffer().getDataType();
145         int offset;
146         for (int k=0; k<bii.getY_dim(); k++){
147             offset = k*bii.getX_dim();
148             switch (dataType)
149             {
150             case 2:
151                 RawDataBufferShort rdbS = (RawDataBufferShort)slice.getRawDataBuffer();
152                 for (int j=0, i=step; i<bii.getX_dim(); i++,j++)
153                 {
```

```
                    rdbS.addElement(rdbS.getElement(offset+i),offset+j);
                }
                for (int i=bii.getX_dim() - step; i<bii.getX_dim() ; i++)
                {
                    rdbS.addElement(Short.MIN_VALUE,offset+i);
                }
                break;
            case 3:
                RawDataBufferUShort rdbU = (RawDataBufferUShort)slice.getRawDataBuffer();
                for (int j=0, i=step; i<bii.getX_dim(); i++,j++)
                {
                    rdbU.addElement(rdbU.getElement(offset+i),offset+j);
                }
                for (int i=bii.getX_dim() - step; i<bii.getX_dim() ; i++)
                {
                    rdbU.addElement(Short.MIN_VALUE,offset+i);
                }
                break;
            default:
                break;
            }
        }
        return slice;
    }

    private MedicalImage2D moveUp(MedicalImage2D slice, short step){
        short dataType = slice.getRawDataBuffer().getDataType();
        int originOffset, destinationOffset;
        switch (dataType)
        {
        case 2:
            RawDataBufferShort rdbS = (RawDataBufferShort)slice.getRawDataBuffer();
            for (int j = 0, i = step ; i<bii.getY_dim(); i++,j++)
            {
                originOffset = i*bii.getX_dim();
                destinationOffset = j*bii.getX_dim();
                for (int k=0 ; k<bii.getX_dim() ; k++ )
                {
                    rdbS.addElement(rdbS.getElement(originOffset+k),destinationOffset+k);
                }
            }
            for (int i=(bii.getY_dim() - step)*bii.getX_dim(); i<bii.getX_dim()*bii.getY_dim() ; i++ )
            {
                rdbS.addElement(Short.MIN_VALUE,i);
            }
            break;
        case 3:
            RawDataBufferUShort rdbU = (RawDataBufferUShort)slice.getRawDataBuffer();
            for (int j = 0, i = step ; i<bii.getY_dim(); i++,j++)
            {
                originOffset = i*bii.getX_dim();
```

```
205                 destinationOffset = j*bii.getX_dim();
206                 for (int k=0 ; k<bii.getX_dim() ; k++ )
207                 {
208                     rdbU.addElement(rdbU.getElement(originOffset+k),destinationOffset+k);
209                 }
210             }
211             for (int i=(bii.getY_dim() - step)*bii.getX_dim(); i<bii.getX_dim()*bii.getY_dim() ; i++ )
212             {
213                 rdbU.addElement(Short.MIN_VALUE,i);
214             }
215             break;
216         default:
217             break;
218         }
219         return slice;
220
221     }
222
223
224     private MedicalImage2D moveDown(MedicalImage2D slice, short step){
225         short dataType = slice.getRawDataBuffer().getDataType();
226         int originOffset, destinationOffset;
227         switch (dataType)
228         {
229         case 2:
230             RawDataBufferShort rdbS = (RawDataBufferShort)slice.getRawDataBuffer();
231             for (int i = bii.getY_dim() - (step+1) , j = bii.getY_dim()-1; i>=0; i--,j-- )
232             {
233                 originOffset = i*bii.getX_dim();
234                 destinationOffset = j*bii.getX_dim();
235                 for (int k=0; k<bii.getX_dim() ; k++ )
236                 {
237                     rdbS.addElement(rdbS.getElement(originOffset+k),destinationOffset+k);
238                 }
239             }
240             for (int i=0; i<step*bii.getX_dim() ; i++ )
241             {
242                 rdbS.addElement(Short.MIN_VALUE,i);
243             }
244             break;
245         case 3:
246             RawDataBufferUShort rdbU = (RawDataBufferUShort)slice.getRawDataBuffer();
247             for (int i = bii.getY_dim() - (step+1) , j = bii.getY_dim()-1 ; i>=0 ; i--,j-- )
248             {
249                 originOffset = i*bii.getX_dim();
250                 destinationOffset = j*bii.getX_dim();
251                 for (int k=0; k<bii.getX_dim() ; k++ )
252                 {
253                     rdbU.addElement(rdbU.getElement(originOffset+k),destinationOffset+k);
254                 }
255             }
```

```
256             for (int i=0; i<step*bii.getX_dim() ; i++ )
257             {
258                 rdbU.addElement(Short.MIN_VALUE,i);
259             }
260             break;
261         default:
262             break;
263         }
264         return slice;
265
266     }
267
268
269
270
271     /*
272     Cette méthode construit une copie de la MedicalImage originale, afin d'annuler les modifications au cas où l'on souhaterait le faire
273     */
274
275     public MedicalImage takeOriginal(){
276         if (img == null)
277         {
278             return null;
279         }
280
281         BasicImageInfo bii = img.getMedicalImageContext().getGeneralInfo().getBasicImageInfo();
282         BasicImageInfo bii2 = new BasicImageInfo();
283         bii2.setX_dim(bii.getX_dim());
284         bii2.setY_dim(bii.getY_dim());
285         bii2.setZ_dim(bii.getZ_dim());
286         bii2.setBits(bii.getBits());
287         bii2.setX_size(bii.getX_size());
288         bii2.setY_size(bii.getY_size());
289         bii2.setZ_size(bii.getZ_size());
290         bii2.setMax(bii.getMax());
291         bii2.setMin(bii.getMin());
292         bii2.setDescription(bii.getDescription());
293         GeneralInfo gi = new GeneralInfo(bii2);
294         MedicalImageContext mic = new MedicalImageContext(gi);
295
296
297
298         if (img.getClass().getName().compareTo("Coreg.Couche1.MedicalImage2D")==0)
299         {
300             MedicalImage2D res = new MedicalImage2D(mic);
301             short dataType = ((MedicalImage2D)img).getRawDataBuffer().getDataType();
302             switch (dataType)
303             {
304             case 2:
305                 RawDataBufferShort rdbS = (RawDataBufferShort)((MedicalImage2D)img).getRawDataBuffer();
306                 RawDataBufferShort rdbS2 = new RawDataBufferShort(rdbS.getValues().length);
```

```
307                 for (int i=0; i<rdbS2.getValues().length; i++)
308                 {
309                     rdbS2.addElement(rdbS.getElement(i),i);
310                 }
311                 res.setRawDataBuffer(rdbS2);
312                 return res;
313             case 3:
314                 RawDataBufferUShort rdbU = (RawDataBufferUShort)((MedicalImage2D)img).getRawDataBuffer();
315                 RawDataBufferUShort rdbU2 = new RawDataBufferUShort(rdbU.getValues().length);
316                 for (int i=0; i<rdbU2.getValues().length; i++)
317                 {
318                     rdbU2.addElement(rdbU.getElement(i),i);
319                 }
320                 res.setRawDataBuffer(rdbU2);
321                 return res;
322             default :
323                 return null;
324             }
325
326         }
327         else{ // MedicalImageStack
328             MedicalImageStack res = new MedicalImageStack(mic,bii.getZ_dim());
329             short dataType;
330             for (int j=0; j<bii.getZ_dim(); j++ )
331             {
332                 MedicalImage2D slice = (MedicalImage2D)((MedicalImageStack)img).getMedicalImage(j);
333                 dataType = slice.getRawDataBuffer().getDataType();
334                 MedicalImage2D sliceRes = new MedicalImage2D(null);
335                 switch (dataType)
336                 {
337                 case 2:
338                     RawDataBufferShort rdbS = (RawDataBufferShort)((MedicalImage2D)slice).getRawDataBuffer();
339                     RawDataBufferShort rdbS2 = new RawDataBufferShort(rdbS.getValues().length);
340                     for (int i=0; i<rdbS2.getValues().length; i++)
341                     {
342                         rdbS2.addElement(rdbS.getElement(i),i);
343                     }
344                     sliceRes.setRawDataBuffer(rdbS2);
345                     res.addMedicalImage(sliceRes,j);
346                     break;
347                 case 3:
348                     RawDataBufferUShort rdbU = (RawDataBufferUShort)((MedicalImage2D)slice).getRawDataBuffer();
349                     RawDataBufferUShort rdbU2 = new RawDataBufferUShort(rdbU.getValues().length);
350                     for (int i=0; i<rdbU2.getValues().length; i++)
351                     {
352                         rdbU2.addElement(rdbU.getElement(i),i);
353                     }
354                     sliceRes.setRawDataBuffer(rdbU2);
355                     res.addMedicalImage(sliceRes,j);
356                     break;
357                 default :
```

```
358                         res = null;
359                     }
360                 }
361                 return res;
362
363             }
364         }
365     }
366
```

```java
package Coreg.Tools;

import Coreg.Couche1.*;
import Coreg.Utils.*;

import java.io.File;

public class Zoomer
{

    private MedicalImage img;

    public Zoomer(MedicalImage img){
        this.img = img;
    }

    static{
        System.loadLibrary("Zoomer");
    }

    public MedicalImage zoom(String[] param){
        ToAIR to = new ToAIR(img, param[0]);
        to.convertToAIRFile();
        doZoomer(param.length, param);
        MedicalImage res = new AIROpener().openImage(param[1]);
        deleteFile(param[0]+".hdr");
        deleteFile(param[0]+".img");
        deleteFile(param[1]+".hdr");
        deleteFile(param[1]+".img");
        return res;
    }

    private void deleteFile(String fileName){
        File file = new File(fileName);
        if (file.exists())
        {
            if (file.delete())
            {
                System.out.println(file.getPath()+" a été correctement effacé");
            }
            else System.out.println("Erreur lors de l'effacement du fichier");

        }
        else System.out.println("Erreur lors de l'effacement du fichier");
    }

    private native void doZoomer(int length, String[] param);
}
```

```java
package Coreg.Utils;

import java.io.File;
import java.io.DataOutputStream;
import java.io.FileOutputStream;

import Coreg.Couche1.*;

public class ToAIR
{
    private MedicalImage img;
    private String pathout; // le pathout ne doit pas contenir d'extension

    private int min,max;

    public ToAIR(MedicalImage img, String pathout){
        this.img = img;
        this.pathout = pathout;
    }

    public void convertToAIRFile(){
            try
            {
                File file = new File(pathout+".hdr");
                file.delete();
                FileOutputStream stream = new FileOutputStream(file);
                DataOutputStream out = new DataOutputStream(stream);
                file.createNewFile();
                writeHeader(out);
                stream.close();
                file = new File(pathout+".img");
                file.delete();
                stream = new FileOutputStream(file);
                out = new DataOutputStream(stream);
                file.createNewFile();
                writeImage(out);
                stream.close();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
    }

    /*writeHeader fonctionne parfaitement !!!!*/

    private void writeHeader(DataOutputStream out){
        BasicImageInfo bii = this.img.getMedicalImageContext().getGeneralInfo().getBasicImageInfo();
        try
        {
            byte[] outtab;
```

```
52              out.writeInt(348); //hdr_size
53              outtab = new byte[28]; // pad1
54              out.write(outtab);
55              outtab = new byte[4]; // extents
56              out.write(outtab);
57              outtab = new byte[2]; //pad2
58              out.write(outtab);
59              outtab = new byte[1]; //regular
60              outtab[0] = (byte)'r';
61              out.write(outtab);
62              outtab = new byte[1]; //pad3
63              out.write(outtab);
64              outtab = new byte[2]; //dims (analyze compatibility)
65              out.write(outtab);
66              out.writeShort(bii.getX_dim());  // xdims
67              out.writeShort(bii.getY_dim()); // ydims
68              out.writeShort(bii.getZ_dim()); // zdims
69              outtab = new byte[2]; // t_dims (analyze compatibility)
70              out.write(outtab);
71              outtab = new byte[20]; //pad4
72              out.write(outtab);
73              outtab = new byte[2]; //datatype (analyze compatibility)
74              out.write(outtab);
75              out.writeShort(bii.getBits()); // bits per pixels
76              outtab = new byte[6]; // pad5
77              out.write(outtab);
78              out.writeFloat(new Double(bii.getX_size()).floatValue()); //x_size
79              out.writeFloat(new Double(bii.getY_size()).floatValue()); //y_size
80              out.writeFloat(new Double(bii.getZ_size()).floatValue()); //z_size
81              outtab = new byte[48]; //pad6
82              out.write(outtab);
83              findMinAndMax();
84              out.writeInt(32767); //max
85              out.writeInt(-32768); //min
86              //out.writeChars(bii.getDescription()); //description
87              outtab = new byte[80];
88              out.write(outtab);
89              outtab = new byte[120]; //pad7
90              out.write(outtab);
91          }
92          catch (Exception e)
93          {
94              e.printStackTrace();
95          }
96
97
98      }
99
100     private void findMinAndMax(){
101
102         if (img.getClass().getName().compareTo("Coreg.Couche1.MedicalImage2D")==0)
```

```
103         {
104             short dataType = ((MedicalImage2D)img).getRawDataBuffer().getDataType();
105             switch (dataType)
106             {
107             case 2 :
108                 min = 32767;
109                 max = -32768;
110                 RawDataBufferShort buffer = (RawDataBufferShort)((MedicalImage2D)img).getRawDataBuffer();
111                 for (int i = 0; i< buffer.getValues().length; i++ )
112                 {
113                     if (buffer.getValues()[i] < min)
114                     {
115                         min = (int)buffer.getValues()[i];
116                     }
117                     if (buffer.getValues()[i] > max)
118                     {
119                         max = (int)buffer.getValues()[i];
120                     }
121                 }
122                 break;
123             case 3 :
124                 min = 32767;
125                 max = -32768;
126                 RawDataBufferUShort bufferU = (RawDataBufferUShort)((MedicalImage2D)img).getRawDataBuffer();
127                 for (int i = 0; i< bufferU.getValues().length; i++ )
128                 {
129                     if (bufferU.getValues()[i] < min)
130                     {
131                         min = (int)bufferU.getValues()[i];
132                     }
133                     if (bufferU.getValues()[i] > max)
134                     {
135                         max = (int)bufferU.getValues()[i];
136                     }
137                 }
138                 System.out.println(min + " " + max);
139                 min = min + 32768;
140                 max = max + 32767;
141                 break;
142             }
143         }
144         else { // MedicalImageStack
145
146             short dataType = ((MedicalImage2D)(((MedicalImageStack)img).getMedicalImage(0))).getRawDataBuffer().getDataType();
147             switch (dataType)
148             {
149             case 2:
150                 min = 32767;
151                 max = -32768;
152                 for (int i = 0; i< ((MedicalImageStack)img).getNumberOfSlices() ; i++)
153                 {
```

```
154                     RawDataBufferShort bufferS = (RawDataBufferShort)((MedicalImage2D)(((MedicalImageStack)img).getMedicalImage(i))).getR
155                     for (int j = 0; j< bufferS.getValues().length; j++ )
156                     {
157                         if (bufferS.getValues()[j] < min)
158                         {
159                             min = (int)bufferS.getValues()[j];
160                         }
161                         if (bufferS.getValues()[j] > max)
162                         {
163                             max = (int)bufferS.getValues()[j];
164                         }
165                     }
166                 }
167                 break;
168 
169             case 3 :
170                 min = 32767;
171                 max = -32768;
172                 for (int i = 0; i< ((MedicalImageStack)img).getNumberOfSlices() ; i++)
173                 {
174                     RawDataBufferUShort bufferU = (RawDataBufferUShort)((MedicalImage2D)(((MedicalImageStack)img).getMedicalImage(i))).ge
175                     for (int j = 0; j< bufferU.getValues().length; j++ )
176                     {
177                         if (bufferU.getValues()[j] < min)
178                         {
179                             min = (int)bufferU.getValues()[j];
180                         }
181                         if (bufferU.getValues()[j] > max)
182                         {
183                             max = (int)bufferU.getValues()[j];
184                         }
185                     }
186                 }
187 
188                 min = min + 32768;
189                 max = max + 32768;
190                 break;
191             }
192         }
193     }
194 
195 
196     /*memo : unsigned short = signed short + 32768*/
197 
198     private void writeImage(DataOutputStream out){
199         /* AIR n'accèpte que les données 16 bits. et Dicom 16 bits signé ou non signés*/
200         int i;
201         if (img.getClass().getName().compareTo("Coreg.Couche1.MedicalImage2D")==0)
202         {
203             System.out.println("MedicalImage2D ok");
204             if (((MedicalImage2D)img).getRawDataBuffer().getDataType()==3/*unsigned short*/){
```

```
205
206                 RawDataBufferUShort buffer = (RawDataBufferUShort)((MedicalImage2D)img).getRawDataBuffer();
207                 for (i=0; i<buffer.getValues().length ;i++ )
208                     {
209                     try
210                     {
211                         out.writeShort(buffer.getElement(i));
212                     }
213                     catch (Exception e)
214                     {
215                         System.out.println("Ecriture du fichier image échoue !");
216                         e.printStackTrace();
217                     }
218
219                 }
220                 return;
221             }
222             if (((MedicalImage2D)img).getRawDataBuffer().getDataType()==2/*signed short*/){
223             }
224
225             RawDataBufferShort buffer = (RawDataBufferShort)((MedicalImage2D)img).getRawDataBuffer();
226             for (i=0; i<buffer.getValues().length ;i++ )
227             {
228                 try
229                 {
230                     out.writeShort(buffer.getElement(i));
231                     if (buffer.getElement(i)>0)
232                     {
233                     }
234
235                 }
236                 catch (Exception e)
237                 {
238                     System.out.println("Ecriture du fichier image échoue !");
239                     e.printStackTrace();
240                 }
241
242             }
243             return;
244         }
245
246         // si on a un stack d'images ...
247
248         if (img.getClass().getName().compareTo("Coreg.Couche1.MedicalImageStack")==0){
249             MedicalImage[] stack = ((MedicalImageStack)img).getStack();
250             MedicalImage2D slice;
251             for (i=0; i<stack.length;i++)
252             {
253                 System.out.println("slice : "+ (i+1));
254                 if (((MedicalImage2D)(stack[i])).getRawDataBuffer().getDataType()==(short)3)
255                 {
```

```
256                     slice = (MedicalImage2D)stack[i];
257                     for (int j = 0; j<((RawDataBufferUShort)slice.getRawDataBuffer()).getValues().length; j++){
258                         try
259                         {
260                             out.writeShort(((RawDataBufferUShort)slice.getRawDataBuffer()).getElement(j));
261                         }
262                         catch (Exception e)
263                         {
264                             e.printStackTrace();
265                         }
266                     }
267                 }
268                 if (((MedicalImage2D)(stack[i])).getRawDataBuffer().getDataType()==(short)2)
269                 {
270                     slice = (MedicalImage2D)stack[i];
271                     for (int j = 0; j<((RawDataBufferShort)slice.getRawDataBuffer()).getValues().length; j++){
272                         try
273                         {
274                             out.writeShort(((RawDataBufferShort)slice.getRawDataBuffer()).getElement(j));
275                         }
276                         catch (Exception e)
277                         {
278                             System.out.println("j'ai foiré dans l'écriture du fichier image !");
279                             e.printStackTrace();
280                         }
281                     }
282                 }
283             }
284
285         }
286     }
287
288     private byte[] int_to_bytetab(int theint){
289         int temp;
290         Integer tempbis;
291         byte[] res = new byte[4];
292         temp = (theint >> 24);
293         tempbis = new Integer(temp);
294         byte byte1 = tempbis.byteValue();
295         res[0] = byte1;
296         temp = (theint << 8);
297         temp = (temp >> 24);
298         tempbis = new Integer(temp);
299         byte byte2 = tempbis.byteValue();
300         res[1] = byte2;
301         temp = (theint << 16);
302         temp = (temp >> 24);
303         tempbis = new Integer(temp);
304         byte byte3 = tempbis.byteValue();
305         res[2] = byte3;
306         temp = (theint << 24);
```

```
307             temp = (temp >> 24);
308             tempbis = new Integer(temp);
309             byte byte4 = tempbis.byteValue();
310             res[3] = byte4;
311             return res;
312         }
313
314         private byte[] short_to_bytetab(short theshort){
315             int temp;
316             Integer tempbis;
317             byte[] res = new byte[2];
318             temp = ( ((int)theshort) >> 8);
319             tempbis = new Integer(temp);
320             byte byte1 = tempbis.byteValue();
321             res[0] = byte1;
322             temp = (((int)theshort) << 24);
323             temp = (temp >> 24);
324             tempbis = new Integer(temp);
325             byte byte2 = tempbis.byteValue();
326             res[1] = byte2;
327             return res;
328         }
329     }
```

```java
package Coreg.Utils;

import java.io.File;
import java.io.DataInputStream;
import java.io.FileInputStream;

import Coreg.Couche1.*;

/**
* <code>AIROpener</code> permet l'ouverture de fichier d'image médicale de type ANALYZE.
*/

public class AIROpener implements CoregOpener {

    private boolean littleEndian = true;

    /**
    *Renvoie une image de type <code>MedicalImage</code> en ouvrant le fichier situé à l'emplacement path
    *@param path le chemin d'accès au fichier à ouvrir
    */

    public MedicalImage openImage(String path){
        MedicalImage img = null;
        String ffile = removeExtension(path,".hdr");
        ffile = removeExtension(ffile,".img");
        File file = new File(ffile+".hdr");
        img = readHeader(file);

        file = new File(ffile+".img");
        if (littleEndian)
        {
            readImage(img,file);
        }
        else {
        readSwappedImage(img, file);
        }

        return img;
    }

/*****************************************HEADER READER*********************************************/

    private MedicalImage readHeader(File file){
        try
        {
        MedicalImage img;
        System.out.println(file.getPath());
        FileInputStream instream = new FileInputStream(file);
        DataInputStream in = new DataInputStream(instream);
        BasicImageInfo bii = new BasicImageInfo();
        int entier;
```

```
52          byte [] ra;
53          char car;
54          short sh;
55          float fl;
56          entier = in.readInt();
57          System.out.println(entier);
58
59          if (entier != 348)
60          {
61              System.out.println("Bad Format : Swapping is needed");
62              instream.close();
63              this.littleEndian = false;
64              img = readSwappedHeader(file);
65              return img;
66          }
67
68          in.skipBytes(38);
69          sh = in.readShort(); //x_dim
70          System.out.println("x_dim : "+sh);
71          bii.setX_dim(sh);
72          sh = in.readShort(); //y_dim
73          System.out.println("y_dim : "+sh);
74          bii.setY_dim(sh);
75          sh = in.readShort(); //z_dim
76          System.out.println("z_dim : "+sh);
77          bii.setZ_dim(sh);
78          sh = in.readShort(); //t_dims
79          ra = new byte[20];
80          entier = in.read(ra); //pad4
81          sh = in.readShort(); //datatype
82          sh = in.readShort(); //bits per pixels
83          bii.setBits(sh);
84          System.out.println("bpp : "+sh);
85          ra = new byte[6];
86          entier = in.read(ra); // pad5
87          fl = in.readFloat();
88          bii.setX_size((double)fl); //x_size
89          System.out.println("x_size : "+fl);
90          fl = in.readFloat();
91          bii.setY_size((double)fl); //y_size
92          System.out.println("y_size : "+fl);
93          fl = in.readFloat();
94          bii.setZ_size((double)fl); //z_size
95          System.out.println("z_size : "+fl);
96          ra = new byte[48];
97          entier = in.read(ra); //pad6;
98          entier = in.readInt(); //max
99          bii.setMax(entier);
100         entier = in.readInt(); //min
101         bii.setMin(entier);
102         ra = new byte[80];
```

```
103         entier = in.read(ra); //description
104         String desc = new String(ra);
105         bii.setDescription(desc);
106
107         GeneralInfo gi = new GeneralInfo(bii);
108         MedicalImageContext mic = new MedicalImageContext(gi);
109
110         if (bii.getZ_dim()>1){
111             img = new MedicalImageStack(mic,bii.getZ_dim());
112         }
113         else{
114             img = new MedicalImage2D(mic);
115         }
116
117
118
119         /*debug : Affichage de toutes les informations de l'objet BasicImageInfo*/
120
121
122         System.out.println(bii.getX_dim());
123         System.out.println(bii.getY_dim());
124         System.out.println(bii.getZ_dim());
125         System.out.println(bii.getBits());
126         System.out.println(bii.getX_size());
127         System.out.println(bii.getY_size());
128         System.out.println(bii.getZ_size());
129         System.out.println(bii.getMax());
130         System.out.println(bii.getMin());
131         System.out.println(bii.getDescription());
132
133         instream.close();
134         /*fin du debug*/
135         return img;
136         }
137         catch (Exception e){
138             e.printStackTrace();
139             return null;
140         }
141
142     }
143
144 /*********************************HEADER READER WITH BYTE SWAPPING*************************************/
145
146         private MedicalImage readSwappedHeader(File file){
147             try
148             {
149                 BasicImageInfo bii = new BasicImageInfo();
150                 FileInputStream instream = new FileInputStream(file);
151                 DataInputStream in = new DataInputStream(instream);
152
153                 byte[] ra = new byte[348];
```

```
154                 in.read(ra);
155                 System.out.println(Tools.swapInt(ra,0));
156                 bii.setX_dim(Tools.swapShort(ra,42));
157                 bii.setY_dim(Tools.swapShort(ra,44));
158                 bii.setZ_dim(Tools.swapShort(ra,46));
159                 bii.setBits(Tools.swapShort(ra,72));
160                 bii.setX_size(Float.intBitsToFloat(Tools.swapInt(ra,80)));
161                 bii.setY_size(Float.intBitsToFloat(Tools.swapInt(ra,84)));
162                 bii.setZ_size(Float.intBitsToFloat(Tools.swapInt(ra,88)));
163                 bii.setMax(Tools.swapInt(ra,140));
164                 bii.setMin(Tools.swapInt(ra,144));
165                 bii.setDescription( new String (ra,148,80));
166                 instream.close();
167 
168                 if (bii.getZ_dim()>1)
169                 {
170                     MedicalImageStack img = new MedicalImageStack(new MedicalImageContext(new GeneralInfo(bii)),bii.getZ_dim());
171                     return img;
172                 }
173                 else {
174                     MedicalImage2D img = new MedicalImage2D(new MedicalImageContext(new GeneralInfo(bii)));
175                     return img;
176                 }
177 
178             }
179             catch (Exception e)
180             {
181                 e.printStackTrace();
182                 return null;
183             }
184 
185         }
186 
187 /****************************************** SWAPPED IMAGE READER ********************************************/
188 
189 /*Dans un premier temps on ne considérera ici que les mesures short*/
190 
191     private MedicalImage readSwappedImage(MedicalImage img, File file){
192         try
193         {
194             System.out.println(file.getPath());
195             FileInputStream instream = new FileInputStream(file);
196             DataInputStream in = new DataInputStream(instream);
197 
198             short x_dim = img.getMedicalImageContext().getGeneralInfo().getBasicImageInfo().getX_dim();
199             short y_dim = img.getMedicalImageContext().getGeneralInfo().getBasicImageInfo().getY_dim();
200             short z_dim = img.getMedicalImageContext().getGeneralInfo().getBasicImageInfo().getZ_dim();
201 
202             byte [] ra = new byte[(int)(x_dim*y_dim*z_dim*2)];
203             int entier = in.read(ra);
204             System.out.println(ra.length);
```

```
205
206             if (img.getClass().getName().compareTo("Coreg.Couche1.MedicalImage2D")==0){
207                 BasicImageInfo bii= img.getMedicalImageContext().getGeneralInfo().getBasicImageInfo();
208                 if (bii.getMax()>32767)
209                 {
210                     RawDataBufferUShort buffer = new RawDataBufferUShort(bii.getX_dim()*bii.getY_dim());
211                     buffer.setDataType((short)3);
212                     for (int i =0 , j = 0;i<bii.getX_dim()*bii.getY_dim(); i++, j+=2 )
213                     {
214                         buffer.addElement(Tools.swapShort(ra,j),i);
215                     //  System.out.println(Tools.swapShort(ra,j));
216                     }
217                     ((MedicalImage2D)img).setRawDataBuffer(buffer);
218                     instream.close();
219                     return img;
220
221                 }
222                 else{
223                     RawDataBufferShort buffer = new RawDataBufferShort(bii.getX_dim()*bii.getY_dim());
224                     buffer.setDataType((short)2);
225                     for (int i =0, j = 0 ;i<bii.getX_dim()*bii.getY_dim(); i++ , j+=2)
226                     {
227                         buffer.addElement(Tools.swapShort(ra,j),i);
228                     //  System.out.println(Tools.swapShort(ra,j));
229                     }
230                     ((MedicalImage2D)img).setRawDataBuffer(buffer);
231                     instream.close();
232                     return img;
233                 }
234             }
235             if (img.getClass().getName().compareTo("Coreg.Couche1.MedicalImageStack")==0){
236                 System.out.println("MedicalImageStack");
237                 BasicImageInfo bii= img.getMedicalImageContext().getGeneralInfo().getBasicImageInfo();
238                 MedicalImage2D slice;
239                 int sliceLength = bii.getX_dim()*bii.getY_dim();
240                 /*Peut etre a modifier*/
241                 if (bii.getMax()>32767){
242                     for (int i=0; i<bii.getZ_dim(); i++ ){
243                         RawDataBufferUShort buffer = new RawDataBufferUShort(bii.getX_dim()*bii.getY_dim());
244                         buffer.setDataType((short)3);
245                         System.out.println("slice "+i);
246                         for (int j=0; j<sliceLength; j++){
247                             buffer.addElement(Tools.swapShort(ra,2*j+i*2*sliceLength),j);
248 //
249 //                          System.out.println(Tools.swapShort(ra,k));
250                         }
251                         slice = new MedicalImage2D(null,buffer);
252                         ((MedicalImageStack)img).addMedicalImage(slice,i);
253                     }
254                     instream.close();
255                     return img;
```

```
256                 }
257                 else{
258                     for (int i=0; i<bii.getZ_dim(); i++ ){
259                         RawDataBufferShort buffer = new RawDataBufferShort(bii.getX_dim()*bii.getY_dim());
260                         buffer.setDataType((short)2);
261                         System.out.println("slice : "+i);
262                         for (int j=0; j<bii.getX_dim()*bii.getY_dim(); j++){
263                             buffer.addElement(Tools.swapShort(ra,2*j+i*2*sliceLength),j);
264                             }
265                         slice = new MedicalImage2D(null,buffer);
266                         ((MedicalImageStack)img).addMedicalImage(slice,i);
267                     }
268                     instream.close();
269                     return img;
270                 }
271
272             }
273             instream.close();
274             return null;
275         }
276         catch (Exception e)
277         {
278             e.printStackTrace();
279             return null;
280         }
281     }
282
283
284
285
286
287 /****************************************** IMAGE READER ******************************************************/
288
289         private MedicalImage readImage(MedicalImage img, File file){
290             try{
291                 System.out.println(file.getPath());
292                 FileInputStream instream = new FileInputStream(file);
293                 DataInputStream in = new DataInputStream(instream);
294
295                 if (img.getClass().getName().compareTo("Coreg.Couche1.MedicalImage2D")==0){
296                     System.out.println("MedicalImage2D");
297                     BasicImageInfo bii= img.getMedicalImageContext().getGeneralInfo().getBasicImageInfo();
298                     if (bii.getMax()>32767)
299                     {
300                         RawDataBufferUShort buffer = new RawDataBufferUShort(bii.getX_dim()*bii.getY_dim());
301                         for (int i =0 ;i<bii.getX_dim()*bii.getY_dim(); i++ )
302                         {
303                             buffer.addElement(in.readShort(),i);
304                             buffer.setDataType((short)3);
305                         }
306                         ((MedicalImage2D)img).setRawDataBuffer(buffer);
```

```
                    return img;

                }
                else{
                    RawDataBufferShort buffer = new RawDataBufferShort(bii.getX_dim()*bii.getY_dim());
                    for (int i =0 ;i<bii.getX_dim()*bii.getY_dim(); i++ )
                    {
                        buffer.addElement(in.readShort(),i);
                        buffer.setDataType((short)2);
                    }
                    ((MedicalImage2D)img).setRawDataBuffer(buffer);
                    return img;
                }



            }
            if (img.getClass().getName().compareTo("Coreg.Couche1.MedicalImageStack")==0){
                System.out.println("MedicalImageStack");
                BasicImageInfo bii= img.getMedicalImageContext().getGeneralInfo().getBasicImageInfo();
                MedicalImage2D slice;
                if (bii.getMax()>32767){
                    for (int i=0; i<bii.getZ_dim(); i++ ){
                        RawDataBufferUShort buffer = new RawDataBufferUShort(bii.getX_dim()*bii.getY_dim());
                        for (int j=0; j<bii.getX_dim()*bii.getY_dim(); j++){
                            buffer.addElement(in.readShort(),j);
                        }
                        slice = new MedicalImage2D(null,buffer);
                        ((MedicalImageStack)img).addMedicalImage(slice,i);
                    }
                    return img;
                }
                else{
                    for (int i=0; i<bii.getZ_dim(); i++ ){
                        RawDataBufferShort buffer = new RawDataBufferShort(bii.getX_dim()*bii.getY_dim());
                        for (int j=0; j<bii.getX_dim()*bii.getY_dim(); j++){
                            buffer.addElement(in.readShort(),j);
                        }
                        slice = new MedicalImage2D(null,buffer);
                        ((MedicalImageStack)img).addMedicalImage(slice,i);
                    }
                    return img;
                }

            }
            return null;
        }
        catch (Exception e)
        {
            e.printStackTrace();
            return null;
```

```
358                 }
359
360             }
361
362
363         private String removeExtension(String file, String extension){
364             if (file.endsWith(extension)){
365                 String res = file.substring(0,file.length()-extension.length());
366                 return res;
367             }
368             return file;
369         }
370
371     }
372
373
374
```

```java
package Coreg.Utils;

import Coreg.Couche1.*;

import emim.em.image.decoder.util.*;
import com.zmed.dicom.*;

import javax.swing.JOptionPane;

/**
*DicomOpener renvoie une structure de type MedicalImage en ouvrant un fichier de type DICOM.
*/

public class DicomOpener implements CoregOpener{

    /**
    *Ouvre un fichier DICOM spécifié à l'emplacement path et renvoie une structure de type MedicalImage
    */
    public MedicalImage openImage(String path){

        //encore à gérer : le fait qu'un fichier soit swappé ou non !

        try{
            DicomImage dimg = new DicomImage(path);
            BasicImageInfo bii = new BasicImageInfo();
            /*Récupérer toutes les informations nécessaires à la création de la BasicImageInfo et appeler le constructeur adéquat*/
            bii.setX_size(dimg.getX_size());
            bii.setY_size(dimg.getY_size());
            bii.setZ_size(dimg.getZ_size());
            bii.setBits((short)dimg.getBitsPerPixels());
            bii.setX_dim((short)dimg.getWidth());
            bii.setY_dim((short)dimg.getHeight());
            bii.setZ_dim((short)dimg.getNbImages());



            GeneralInfo gi = new GeneralInfo(bii);
            MedicalImageContext mic = new MedicalImageContext(gi);
            int nbre_slices = dimg.getNbImages();
            int largeur = dimg.getWidth();
            int hauteur = dimg.getHeight();
            MedicalImage2D img;
            RawDataBuffer buffer;
            byte tab[];
            short value;
            int k;
            if (nbre_slices>1){ //MedicalImageStack //suppose little endian
                MedicalImageStack mis = new MedicalImageStack(mic,nbre_slices);
                if (dimg.getFileType()==DicomImage.GRAY16_UNSIGNED)
                {
                    for (int i = 0; i<nbre_slices; i++){
```

```
52                      buffer = new RawDataBufferUShort(hauteur*largeur);
53                          for (int j = 0; j<hauteur*largeur; j++){
54                              k = 2*j + i*2*hauteur*largeur;
55
56                              ((RawDataBufferUShort)buffer).addElement((short)((((dimg.pixelsData[k+1]&0xff)<<8) |
                                (dimg.pixelsData[k]&0xff))+32768),j);
57                          }
58                          img = new MedicalImage2D(null,buffer);
59                          mis.addMedicalImage(img,i);
60                      }
61                      return mis;
62                  }
63                  else { /*GRAY16_SIGNED*/
64                      for (int i = 0; i<nbre_slices; i++){
65                      buffer = new RawDataBufferShort(hauteur*largeur);
66
67                          for (int j = 0; j<hauteur*largeur; j++){
68                              k = 2*j + i*2*hauteur*largeur;
69                              ((RawDataBufferShort)buffer).addElement((short)((((dimg.pixelsData[k+1]&0xff)<<8) |
                                (dimg.pixelsData[k]&0xff))),j);
70                          }
71                          img = new MedicalImage2D(null,buffer);
72                          mis.addMedicalImage(img,i);
73                      }
74                      return mis;
75                  }
76              }
77              else { //une seule image 2D
78
79                  if (dimg.getFileType()==DicomImage.GRAY16_UNSIGNED)
80                  {
81                      buffer = new RawDataBufferUShort(hauteur*largeur);
82                      for (int j = 0; j<hauteur*largeur; j++){
83                          k = 2*j;
84
85                          ((RawDataBufferUShort)buffer).addElement((short)((((dimg.pixelsData[k+1]&0xff)<<8) |
                            (dimg.pixelsData[k]&0xff))+32768),j);
86
87                      }
88                      img = new MedicalImage2D(mic,buffer);
89                      return img;
90                  }
91
92                  else {
93                      buffer = new RawDataBufferShort(hauteur*largeur);
94                      for (int j = 0; j<hauteur*largeur; j++){
95
96                          k = 2*j;
97                          ((RawDataBufferShort)buffer).addElement((short)(((dimg.pixelsData[k+1]&0xff)<<8) |
                            (dimg.pixelsData[k]&0xff)),j);
98                      }
```

```
 99                      img = new MedicalImage2D(mic,buffer);
100                      return img;
101                  }
102
103              }
104
105          }
106          catch(Exception e){
107              JOptionPane.showMessageDialog(null,"Erreur lors de l'ouverture du fichier : format innattendu", "erreur",
                 JOptionPane.ERROR_MESSAGE);
108              e.printStackTrace();
109              return null;
110          }
111      }
112
113      private byte[] swapDicomImage(byte[] buffer){
114          /*on sait que le buffer est d'une taille multiple de 2*/
115          for (int i=0 ; i<buffer.length ; i+=2 )
116          {
117              byte temp = buffer[i];
118              buffer[i] = buffer[i+1];
119              buffer[i+1] = temp;
120          }
121          return buffer;
122      }
123  }
```

```java
package Coreg.Utils;

import com.zmed.dicom.*;

/**
*<code>DICOMDataObjectOperations</code> permet d'effectuer des modifications sur des images de type DICOM, tel qu'ouvrir un fichier DICOM, modifier des éléments de son header, la valeur des pixels, sauver l'image modifiée au format DICOM.
*/

public class DICOMDataObjectOperations
{

    private String filename;
    DICOMClientServer pdu;
    DICOMDataObject ddo;

    /**
    *Permet de charger en mémoire une image de type DICOM afin d'effectuer des modifications sur celles-ci. Pour plus d'information, consulter la doc sur le package zdicom
    */
    public DICOMDataObjectOperations(String filename) throws Exception{
        this.filename = filename;
        pdu = new DICOMClientServer(new DICOMMessage());
        ddo = pdu.loadDICOMDataObject(filename);
    }

    /**
    *Spécifie la taille des voxels
    *@param x_size la largeur des voxels
    *@param y_size la hauteur des voxels
    *@param z_size la profondeur des voxels
    */
    public void setVoxelSize(double x_size, double y_size, double z_size/*spacing between slices*/){
        String str = new String(Double.toString(x_size)+"\\"+Double.toString(y_size));
        ddo.setS(RTC.DD_PixelSpacing,str);
        ddo.setS(RTC.DD_SpacingBetweenSlices,Double.toString(z_size));
    }

    /**
    *Spécifie le nombre de slice de l'image
    *@param nbre le nombre de slices
    */
    public void setNumberOfSlices(int nbre){
        String nbreSlices = Integer.toString(nbre);
        ddo.setS(RTC.DD_NumberOfFrames,nbreSlices);
    }

    /**
    *Spécifie la résolution de l'image en largeur et en profondeur
    *@param x_dim la largeur de l'image en nombre de voxels
    *@param y_dim la hauteur de l'image en nombre de voxels
```

```
50      */
51
52      public void setDims(int x_dim, int y_dim){
53          ddo.setI(RTC.DD_Rows,y_dim);
54          ddo.setI(RTC.DD_Columns, x_dim);
55      }
56
57      /**
58      *Spécifie la représentation du voxel. 1 est relatif aux <code>Signed Short</code> et 0 aux <code>Unsigned Short</code>
59      *@param x la valeur de la représentation du pixel (usuellement 0 ou 1)
60      */
61      public void setPixelRepresentation(int x){
62          ddo.setI(RTC.DD_PixelRepresentation,x);
63      }
64
65
66      /**
67      *Spécifie l'ensemble des pixels de l'image sous la forme d'un tableau de bytes
68      *@param pixs le tableau de byte représentant le tableau de voxels
69      */
70      public void setPixelsData(byte[] pixs){
71          VR pixData = new VR();
72          pixData = ddo.getVR(0x7fe0,0x0010);
73          pixData.data = pixs;
74      }
75
76      /**
77      *Permet de sauver l'image référencée par <code>DICOMDataObjectOperations</code> sur le disque à l'emplacement outfile
78      *@param outfile le chemin où sauver le fichier
79      */
80      public void saveDICOM(String outfile) throws Exception{
81          pdu.saveDICOMDataObject(outfile,ddo);
82      }
83
84      /**
85      *Spécifie le nombre de bits surlequel l'image est encodé
86      *@param bits le nombre de bits surlequel chaque voxel de l'image est encodé
87      */
88      public void setBits(short bits){
89          ddo.setI(RTC.DD_BitsAllocated,bits);
90      }
91
92  }
93
```